Fine-Grained Clothing Image Retrieval with Faster R-CNN

By

Ong Tun Ying

TARC

TUNKU ABDUL RAHMAN
UNIVERSITY COLLEGE

FACULTY OF COMPUTING AND
INFORMATION TECHNOLOGY

TUNKU ABDUL RAHMAN UNIVERSITY COLLEGE
KUALA LUMPUR

ACADEMIC YEAR
2019/20

Fine-Grained Clothing Image Retrieval with Faster R-CNN

By

Ong Tun Ying

Supervisor: Dr. Lim Khai Yin

A project report submitted to the
Faculty of Computing and Information Technology
in partial fulfillment of the requirement for the
Bachelor of Computer Science (Honours)

Department of Computer Science and Embedded Systems
Faculty of Computing and Information Technology
Tunku Abdul Rahman University College
Kuala Lumpur

2019/20

Copyright by Tunku Abdul Rahman University College.
All rights reserved. No part of this project documentation may be reproduced, stored in retrieval system, or transmitted in any form or by any means without prior permission of Tunku Abdul Rahman University College.

Declaration

The project submitted herewith is a result of my own efforts in totality and in every aspect of the project works. All information that has been obtained from other sources had been fully acknowledged. I understand that any plagiarism, cheating or collusion or any sorts constitutes a breach of TAR University College rules and regulations and would be subjected to disciplinary actions.

_____

Ong Tun Ying

Bachelor of Computer Science (Honours) in Software Engineering

ID: 18WMR08499

Abstract

It is clear that image retrieval based application is still considerably niche in the fashion and clothing industry. Many have attempted to develop frameworks and methods to build such applications, while they succeed to make image retrieval out of fashion images, the lack of performance and accuracy causes them to be unsuitable for real-world usage. This research is another such attempt to solve the existing problems of lack of performance and accuracy. This research covers topics such as pattern recognition, convolutional neural networks (CNN). CNN architectures, and methodologies to solve the underlying problems such as choosing the correct dataset, dataset conversion, object detection, and transfer learning.

Acknowledgement

My utmost gratitude is given to my research supervisor, Dr. Lim Khai Yin for her dedication and her effort to get involved in assisting in this project throughout the process until completion. The successful completion of the project would not be possible without her.

I also owe my gratitude and thanks to my classmates, whom have been supportive and kind to me when I need their assistance and advice. Overcoming the challenges and hardships for this project required more than just academic support. That being said, I have countless people to thank for listening and motivating me relentlessly over the past three years. I cannot begin to express my gratitude and appreciation for their friendship and support.

Last but not least, without my dearest family, none of these would be possible. To my parents, thank you for always being there for all my ups and downs for the past three years. Thank you for always providing me with a safe and peaceful home that I could return to after each exhausting semester. I am truly grateful for the unconditional love given to me and I would never take it for granted.

Table of Contents

Chapter 1

Introduction

1 Introduction

In 2019, fashion and clothing image retrieval remains to be an advanced image processing framework which has high potential in serving needs such as computer aided fashion design (fashion recommendation system), customer profile analysis, and context-aided people identification. However, it is rare to find a framework which can recognize fashion categories and attributes through real-time surveillance cameras due the needs to track the human subjects, segmentation of clothing, and creating a library of categories and attributes for the clothings (Yang and Yu, 2011). Using DeepFashion's dataset that comes with 50 established categories and 1000 attributes to be used (Liu et al., 2016), along with TensorFlow's (Abadi et al., 2016) built-in object detection and tracking models like Faster R-CNN (Szegedy et al., 2015), a deep learning framework is built for fine-grained clothing image retrieval. This framework could potentially benefit fashion companies and ecommerce companies as well as any fashion enthusiast worldwide in acquiring data of fashion trends (Wired, 2019), to answer simple questions like: "What are the most favored colors for women's fashion in Malaysia?" without even asking anyone (Innovation Enterprise, n.d.).

1.1 Problem Statement

Though many researches have been performed to harvest the potential of fashion and clothing recognition, most researches focus on the reliability and performance of the deep learning models (Yang and Yu, 2011). Notwithstanding the importance of such, very few developments have been done in the fashion field to implement and to apply the deep learning models. However, the primary reasons for this lack of development are not due to the lack of interest, but rather due to many other sub-problems that are challenging to overcome in building such a system.

First, such a framework might involve sub-problems such as face detection and/or body tracking, human figure (poselets or body parts) or clothing (landmarks) segmentation, and effective clothing representations.

Second, the discerning of various subtle differences among fashion categories and attributes requires considerable computations to be considered well-achieved.

1.2 Research Objectives

- To explore every important topic that is needed to build a fine-grained clothing image retrieval framework.
- To analyze the topics, and choose the most preferred methods to build to framework. The framework needs to be able to solve problems mentioned in Section 1.1, which are:
    - ➢ To be able to detect/track objects based on an input, which can be still images or motion images.

    - ➢ To be able to retrieve clothing information based on fashion categories and attributes from the images.
- Design and build the framework.
- Evaluate the results.

1.3 Research Scope

In making sure that the proposed framework is able to solve real-world problems, as well as being relevant in the real-world, a scope is defined for this research, the topics which have been explored are:

- Theories related to pattern recognition.
- The artificial neural network.
- CNNs and publicised architectures based on it.
- Every possible methodologies and considerations in building the proposed framework, which includes:
    - ➢ The dataset that is needed.
    - ➢ The CNN architecture that is being referred to.
    - ➢ The existing frameworks/tools needed to build the framework.
    - ➢ Methods to evaluate the results.
- The evaluation of the results.
- Comparisons with other models.
- Discussions of achievements, issues, solutions, limitations and future improvements.

Chapter 2

Research Background

2 Literature Review

In this chapter, relevant researches related to the project will be presented. Topics such as pattern recognition, neural networks, convolutional neural networks (CNN), and CNN architectures shall be discussed in detail in this chapter.

2.1 Pattern Recognition

The discipline of pattern recognition is concerned with the automatic discovery of fundamental structure in the input objects based on preexisting knowledge or statistical information extracted from the structure through the use of machine intelligence (Chakrabarty, 2010) (Howard, 2007).

In the past, two prominent approaches to pattern recognition are statistical, and structural. Today, artificial neural networks has provided an alternative - neural pattern recognition.

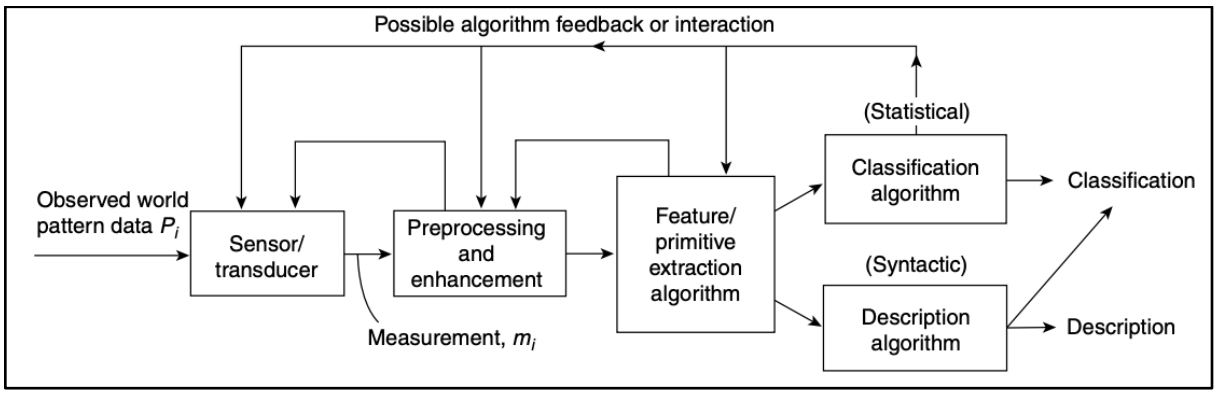


Figure 2.1 illustrates a generic pattern recognition system structure that is applicable to all three of the aforementioned approaches. Notice that it consists of a sensor(s), which in some cases, pre-classified data may also be used and are usually called the 'training data' (Schalkoff, 2007).

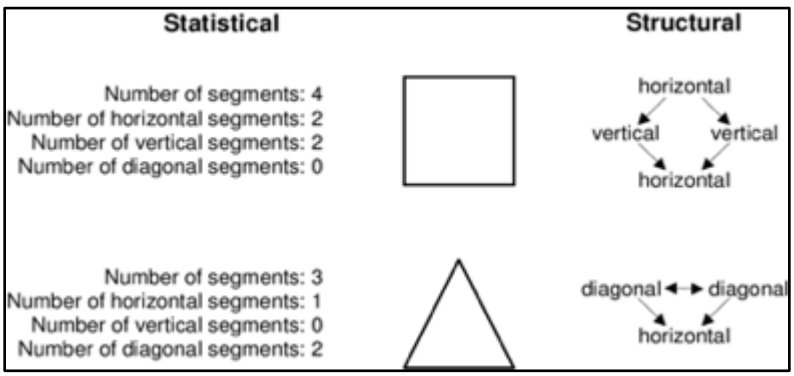


Figure 2.2: The statistical and structural approaches to pattern recognition applied to a common identification problem. The goal is to discriminate between the square and the triangle (Olszewski R. T. 2001). .

2.1.1 Patterns and Features

Patterns can be a set of measurements or observations, usually represented in vector notation. Whereas features are the measurements extracted from the patterns. Features may be symbolic, numeric, or both. Color is an example of a symbolic feature, whereas weight is an example of a numerical feature. Feature extraction algorithms are used to extract features from patterns. Extracted features may still contain 'noise' or unwanted measurements.

2.1.2 Statistical Approach to Pattern Recognition

The statistical approach defines that the measurements taken from N features are represented in N-dimensional pattern space as one point. The portion of the pattern space defines the principle of the classification into subspaces, each corresponding to a specific pattern class.Sometimes, the rejection of the classification of some patterns can lead to desirable outcomes.

Parametric classification methods, discriminant functions, clustering analysis, and fuzzy set reasoning are some techniques that use the statistical approach. When there is no prior knowledge of the information, the use of fuzzy set reasoning creates an alternative to the probabilistic approach (Zadeh, 1965).

However, the techniques above cause optimization problems. Besides, the quantification of the contribution of a particular feature towards the accuracy of classification also creates ambiguity. This together with the indeterminacy in the statically inter-relationship between features, causes their specification to become an exercise in educated guessing (Nandhakumar and Aggarwal, 1985). In general, all pattern features cannot be represented exactly in mathematical expressions and the evaluation of the effectiveness of feature ordering largely depends on human subjective decisions.

2.1.3 Structural Approach to Pattern Recognition

When one talks about the structural approach to pattern recognition, one usually means that it is a recursive process of defining complex patterns to their less complex form (Kepka, 1994). The results of this simplification process is a set of pattern primitives (simplest forms of many sub-patterns) and their relationships.

Specified or given set of grammar would be used for syntax analysis where to grammar would be used to determine whether the pattern representation is correct. If the pattern cannot be parsed in any possible classes of patterns, then the pattern shall be rejected. The syntax analysis would produce a complete syntactic description (or a parsing tree), which is either constructed from top to bottom or from bottom to top.

Furthermore, one should have the knowledge about the costs between the complexity of the regnizer and the defining power of the language used for the grammar, as the cost must also be defined. When it comes to multidimensional patterns – 2D and 3D images, high-dimensional pattern grammars are used such as tree grammars, plex grammars, web grammar, etc.

The limitations to this approach can be seen when one finds out that it is not capable in handling numerical semantic information for recognition. Moreover, when a pattern can be generated by more than one pattern grammar, ambiguity becomes an issue.

2.2 Neural Networks

Neural networks are algorithms that are designed to do pattern recognition. They are modeled loosely after the human brain. Sensory data are interpreted through labeling or clustering on raw input, hence the patterns they are able to recognize are usually numerical, or real-world data that are contained in vectros, such as images, sound, text or time series (Nicholson, n.d.).

A standard neural network (NN) consists of many connected processors called neurons, each producing a sequence of real-valued activations. Neurons such as input neurons get activated through sensors that are set to perceive the environment, while the other neurons get activated through weighted connections from other activated neurons (Schmidhuber, 2015).

McCulloch and Pitts' made the first breakthrough in Artificial Neural Network (ANN) in the 1940s (McCulloch and Pitts, 1990). It was followed by Rosenblatt's perceptron convergence theorem and Minsky and Papert's work which showed the limitations of a simple perceptron in the 1960s (Nievergelt, 1969) Hopfield's energy approach in 1982 (Hopfield, 1982) and the Werbos' proposal of the back-propagation learning algorithm for multilayer perceptron's (multilayer feedforward networks) also helped to renew interest in the field of ANN (Werbos, 1974).

2.2.1 Neural Network Architectures

ANNs can be considered as weighted directed graphs in which the neurons are nodes and directed edges (each carrying a weight) are the connections between the neuron outputs and the neuron inputs.

There are generally two categories of ANNs:

- Feed-forward networks, graph has no loops
- Feedback networks, loops occur in graph

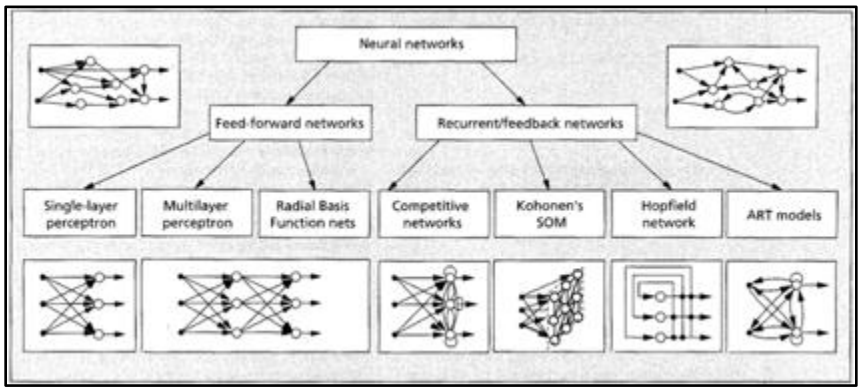


Figure 2.4: An array of feed-forward and feedback network architectures (Jain et al., 1996).

Different architecture yields different network behaviors. In general, feed-forward networks produce only one set of output values rather than a sequence values, and therefore they are considered static. Feed-forward networks require less memory as their response to an input is independent of the previous network state.

However, recurrent, or feedback networks are considered dynamic systems. This can be explained by the computation of each neuron output when each of the new input pattern is presented, and due to the paths being feedback paths, the inputs of each neuron are modified each time they are passed, hence leading the network to a new state (Jain et al., 1996).

There are three main learning paradigms: supervised, unsupervised, and hybrid.

In supervised learning, the model is provided with a correct answer (output) for every input pattern. Weights are determined to allow the model to produce answers approximate to correct answers. Reinforcement learning is a variant of supervised learning in which the network is provided with only a critique on the correctness of network outputs, not the correct answers themselves.

In contrast, unsupervised learning does not require a correct answer associated with each input pattern in the training data set. It explores the underlying structure in the data, or relationships between patterns in the data, and organizes patterns into categories from these matchings.

Hybrid learning combines supervised and unsupervised learning. Part of the weights are usually determined through supervised learning, while the others are obtained through unsupervised learning.

Most ANNs consists of three layers of processing units: the input layer, hidden layer and outer layer. The units in the input layer are connected to the units in hidden layer, which are then connected to units in the output layer (Chakrabarty, 2010). The figure below showcases the interconnections between the aforementioned layers.

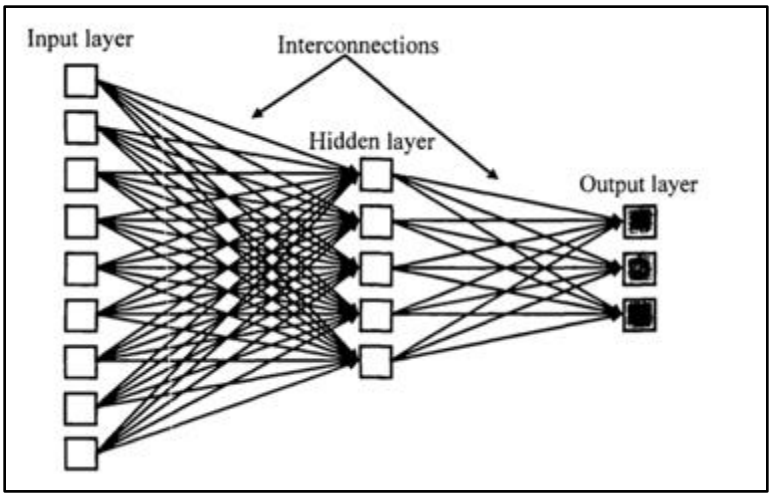


Figure 2.5: An artificial neural network with three layers of processing units (Zakaria et al., 2014).

2.2.2 Training Process

The training process of a neural network consists of defining a cost function and using gradient descent optimization to minimize it.

Mean Squared Error (MSE) is used as the cost function. It can be equated to:

MSE = Sum [(Prediction - Actual)^2] * (1 / number of observations)

The MSE works as an estimator (of a procedure for estimating an unobserved quantity) measuring the average of the squares of the errors — that is, the average squared difference between the estimated values and what is estimated. MSE is a risk function, corresponding to the expected value of the squared error loss. MSE results are mostly positive (and not zero) due to randomness or because the estimator does not account for information that could produce a more accurate estimate (Yiu, 2019).

Gradient Descent is the vector of the gradient in a function whose elements are its partial derivatives with respect to each parameter. For example, in attempt to minimize a cost function, C(B0, B1), with just two changeable parameters, B0 and B1, the gradient would be:

Gradient of C(B0, B1) = [[dC/dB0], [dC/dB1]]

With this, each element of the gradient can tell how the cost function would change if a small change to that particular parameter is applied. This makes tweaking the network with approximate values possible.

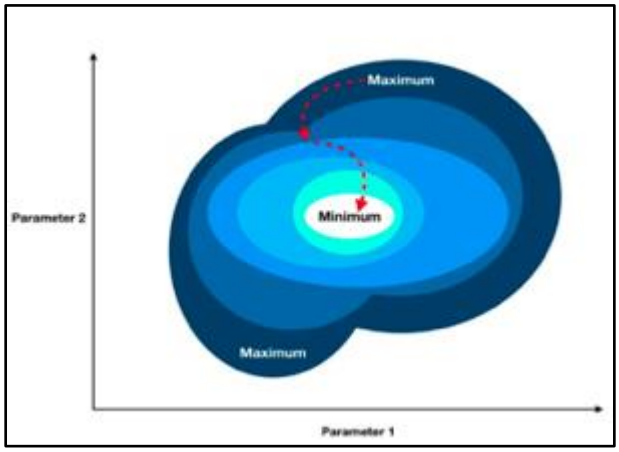


Figure 2.6: Illustration of gradient descent (Yiu, 2019).

To summarize, the small changes can be computed through these steps:

1. Compute the gradient of the current coordinates.
2. Modify each parameter by an amount proportional to its gradient element and in the opposite direction of its gradient element. For example, if the partial derivative of the cost function with respect to B0 is positive but small and the partial derivative with respect to B1 is negative and large, then B0 shall be decreased by a tiny amount and increase B1 by a large amount to lower the cost function.
3. Recompute the gradient using the newly tweaked parameter values and repeat the previous steps until the minimum value is reached.

2.3 Convolutional Neural Network (CNN)

In 1989, LeCun invented the convolution of the neural network - LeNet, and its use for digital identification (LeCun et al., 1989). Then in 2012, in response to questions regarding deep learning, Hinton and his students applied deep learning to ImageNet (the largest database in image recognition), and achieved remarkable results (Krizhevsky et al., 2012). CNNs are a variance of neural networks, hierarchical neural network for recognition, particularly images. It works by using significant processes, such as Gradient Descent (Bottou, 2010) and Backpropagation. Furthermore, a typical CNN consists of a sequence of layers, and every layer transform one volume of activations to another through the use of specific functions. These layers consists of the beginning layer, convolution layer, and the output layer. The layers between them are called hidden layers. Then main purpose of convolution layer is to extract image features, then drive them into the hidden layers for computing, and output the results via output layer. Layers among hidden layers usually, such as pooling layers (sub-sampling layers) are partially connected, while the output layers are fully connected (Zhou, 2018). There are several architectures within CNN that are commonly used: AlexNet, LeNet, Inception-v1 (GoogleNet), ResNet, and VGGNet.

2.3.1 Pooling Layers

A pooling layer is added in-between successive Convolutional layers in a CNN architecture. Its function is to gradually reduce the spatial size of the representation to reduce the number of parameters and computation in the network. The Pooling Layer operates independently on every depth slice of the input and resizes it spatially, using the MAX operation (CS231n, n.d.). The most common form is a pooling layer with filters of size 2x2 applied with a stride of 2 down samples every depth slice in the input by 2 along both width and height, discarding 75% (3/4) of the activations. Every MAX operation would in this case be taking a max over 4 numbers (little 2x2 region in some depth slice). However, depth dimension remains unchanged. It works by:

- Accepting a volume of size W1×H1×D1
- Requires two hyperparameters: The spatial extent F, The stride S
- Produces a volume of size W2×H2×D2 where:
    - $W2=(W1-F)/S+1$
    - $H2=(H1-F)/S+1$
    - D2=D1

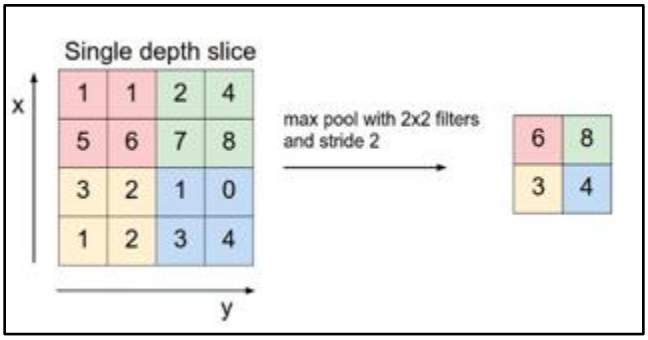


Figure 2.7: Max pooling being performed on a single depth slice; each max is taken over 4 numbers (little 2x2 square) (CS231n, n.d.)

2.3.2 Process

The convolutional layer consists of a set of filters. Each filter is smaller than the input data and the type of multiplication applied between a filter-sized patch of the input and the filter is a dot product resulting in a single value. This method allows the same filter (set of weights) to be multiplied by the input array multiple times at different points on the input.

Specifically, the filter is applied systematically to each overlapping part or filter-sized patch of the input data, left to right, top to bottom. This results in a two-dimensional array of output values that represent a filtering of the input called the "feature map" (Brownlee, 2019).

Successive computational layers alternate between convolutional layers and sampling layers, as spatial resolution decreases in each convolutional layer or sampling layer, the number of feature mappings increases compared to the corresponding previous layer (Zhou, 2018).

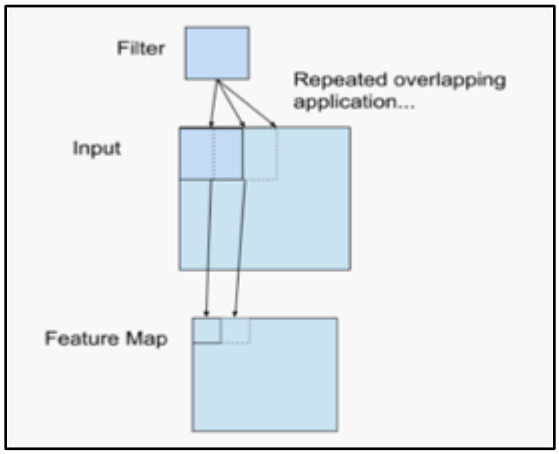


Figure 2.8: Example of a Filter Applied to a Two-Dimensional Input to Create a Feature Map (Brownlee, 2019).

For example, the input layer consists of 32 × 32 sensing nodes, receiving the original image. The calculation then alternates between convolution and sub-sampling, which stated as follows:

The first hidden layer is convoluted, which consists of eight feature maps, each feature map consists of 28 × 28 neurons, each neuron specifies a 5 ×5 acceptance domain; The second hidden layer implements sub-sampling and local averaging. It is also composed of eight feature maps, but each feature map consists of

14 × 14 neurons. Each neuron has a 2 × 2 acceptance field, a training coefficient, a training bias, and a sigmoid activation function. The training factor and the bias control the operating point of the neuron. The third hidden layer is the second convolution, which consists of 20 feature maps, each consisting of 10 × 10 neurons. Each neuron in the hidden layer may have a synaptic connection to several feature maps of the next hidden layer, which operates in a similar manner to the first convolution layer. The fourth hidden layer performs the second sub-sampling and local average calculation. It consists of 20 feature maps, but each feature map consists of 5 x 5 neurons, which operate in a similar manner to the first sample. The fifth hidden layer implements the final stage of convolution, which consists of 120 neurons, each of which specifies a 5 × 5 accepted domains. Finally, a full connection layer, get the output vector.

2.3.3 Backpropagation

Backpropagation is an expression for the partial derivative ∂C/∂w of the cost function C with respect to any weight w (or bias b) in the network (Nielsen, 2019). The expression below is used to determine how quickly the cost changes when the weights and biases are changed.

Definition of the backpropagation error:

$$\frac{\partial E}{\partial b} = \frac{\partial E}{\partial u} \times \frac{\partial u}{\partial b}$$

2.4 CNN Architectures

From 2010 to 2017, The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) evaluates algorithms for object detection and image classification at large scale (Russakovsky et al., 2015). One of their high level of motivation is to allow researchers to compare progress in detection across a wider variety of objects -- taking advantage of the quite expensive labeling effort. Since then. many architectures and methods on image recognition are born, most of them are based on CNN.

2.4.1 AlexNet

In 2012, an architecture used in a research paper titled, “ImageNet Classification with Deep Convolutional Neural Networks” became known as AlexNet due to the first author named Alex Krizhevsky (Krizhevsky et al., 2012). The paper used its proposed framework on CNN to get a Top-5 error rate (rate of not finding the true label of a given image among its top 5 predictions) of 15.3%, which was the lowest rate of that time. AlexNet was the winning entry in ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2012. It solved the problem of image classification where the input is an image of one of 1000 different classes and the output is a vector of 1000 numbers (Nayak, 2018).

AlexNet utilizes an architecture that is much larger than previous CNNs used in computer vision. It consists of 5 Convolutional Layers and 3 Fully Connected Layers with over 60 million parameters and 650,000 neurons.

The first two Convolutional layers are followed by the Overlapping Max Pooling layers. The third, fourth and fifth convolutional layers are connected directly. The fifth convolutional layer is followed by an Overlapping Max Pooling layer, the output of which goes into a series of two fully connected layers. The second fully connected layer feeds into a SoftMax classifier with 1000 class labels. Lastly, ReLU (Rectified Linear Unit) nonlinearity is applied to the first and second convolution layer followed by a local normalization step before pooling.

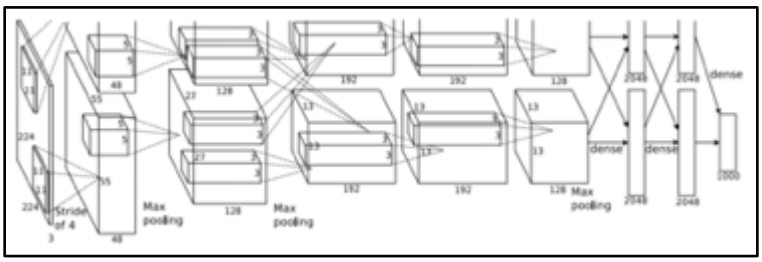


Figure 2.9: AlexNet architecture showing the delegation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom (Krizhevsky et al., 2012).

Max Pooling Layers down sample the width and height of the tensors, maintaining the depth. Overlapping Max Pool layers are similar to the Max Pool layers, except the adjacent windows over which the max is computed overlap each other. The authors used pooling windows of size 3×3 with a stride of 2 between the adjacent windows. This overlapping nature of pooling helped reduce the top-1 error rate by 0.4% and top-5 error rate by 0.3% respectively when compared to using non-overlapping pooling windows of size 2×2 with a stride of 2 that would give the same output dimensions.

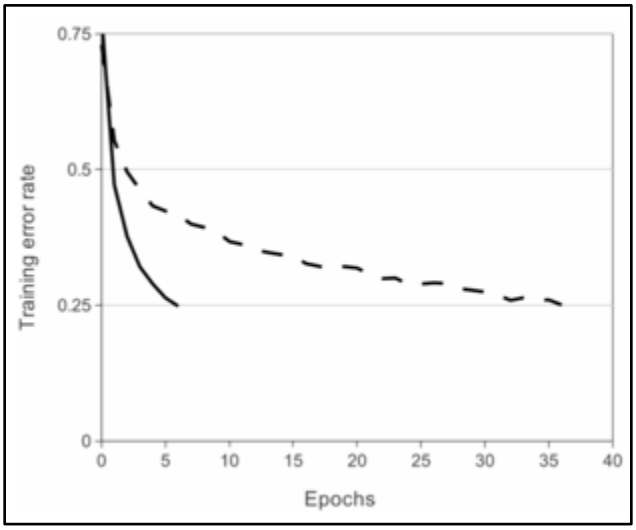


Figure 2.10: A four-layer CNN with ReLUs (solid line) training error rate on CIFAR-10 against tanh neurons (dashed line) (Krizhevsky et al., 2012).

The use of ReLu Nonlinearity allowed deep CNNs to be trained much faster than using the saturating activation functions like tanh or sigmoid. Figure 2.10 shows through the use of ReLUs(solid curve), AlexNet could achieve a 25% training error rate six times faster than an equivalent network using tanh(dotted curve).

With 60 million parameters, the author had unique ways to combat overfitting on image data. One of which is to artificially enlarge the dataset using label-preserving transformations. Two distinct forms of data augmentation were employed, both of which allowed transformed images to be produced from the original images with minimal computation. The first form of consists of generating image translations and horizontal reflections through extracting g random 224 × 224 patches (and their horizontal reflections) from the 256×256 images and training these images on the network. The second method is to alter the intensities of the RGB channels in training images. The results of the model are summarized in the figure below:

Model	Top-1	Top-5
Sparse coding [2]	*47.1%*	*28.2%*
SIFT + FVs [24]	*45.7%*	*25.7%*
CNN	**37.5%**	**17.0%**

Table 2.1: Comparison of results on ILSVRC2010 test set, CNN using AlexNet framework (Krizhevsky et al., 2012).

2.4.2 VGGNet

VGGNet is invented by VGG (Visual Geometry Group) from the University of Oxford in 2014. It won the 1st runner-up prize of the ILSVRC (ImageNet Large Scale Visual Recognition Competition) 2014 in the classification task and won first place in the localization task category (Simonyan and Zisserman, 2015).

VGGNet utilizes a smaller filter size of 3x3 as opposed to the usual 11x11 in AlexNet. However, by using 3 layers of 3x3 filters, it effectively covers an area of 7x7. This also means that the number of parameters is fewer. Suppose there is only 1 filter per layer and 1 layer at input:

- 1 layer of 11×11 filter, number of parameters = 11×11=121
- 5 layers of 3×3 filters, number of parameters = 3×3×5=45

This results in the total number of parameters to be reduced by 63%. On a larger network, fewer parameters to be learnt produces a faster convergence and reduces the overfitting problem prevalent in older CNN models such as AlexNet.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure 2.11: VGGNet configurations (In columns). The depth of the configurations increases from the left (A) to the right (E), as more layers are added. The incorporation of 1 × 1 conv. layers (configuration C, Table 1) is a way to increase the nonlinearity of the decision function (Simonyan and Zisserman, 2015).

The architecture utilizes a stack of convolutional layers (which has a different depth in different architectures) is followed by three Fully-Connected (FC) layers: the first two have 4096 channels each, the third performs 1000- way ILSVRC classification and thus contains 1000 channels (one for each class). The final layer is the soft-max layer.

Besides this, all hidden layers are equipped with ReLu non-linearly (Tsang, 2018). Through Figure 2.12, the macro architecture of configuration D of the network (VGG-16) is visualized.

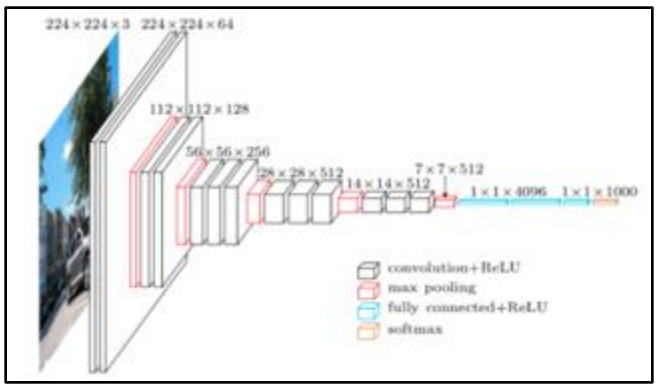


Figure 2.12: Macro Architecture of VGG-16 (Tsang, 2018).

VGG-16 managed to obtained a 9.4% error rate compared the VGG-11 rate of 10.4%, which means the additional three 1×1 conv layer helps improve the classification accuracy. Moreover, it proves that the network improves by adding the number of layers.

One method VGGNet applies to reduce the error rate is to train the network at various scales. For example, in single-scale training an image is scaled with smaller-size equal to 256 or 384, i.e. S=256 or 384, whereas in multi-scale training an image is scaled with smaller-size equal to a range from 256 to 512, i.e. S=[256;512]. Therefore, the range of S is large with multi-scale training. Results shown a reduction in the error rate of VGG-16 from 8.8%/8.7% to 8.1%.

2.4.3 ResNet

ResNet (He et al. 2015), short for Residual Networks is a classic neural network used as a backbone for many computer vision tasks. This model won the ImageNet challenge in 2015. The fundamental breakthrough with allowed training extremely deep neural networks with 150+layers successfully. Prior to ResNet, training very deep neural networks was difficult due to the eventual saturation and the notorious problem of vanishing gradients. The details within the architecture and process behind ResNet will be further explained below.

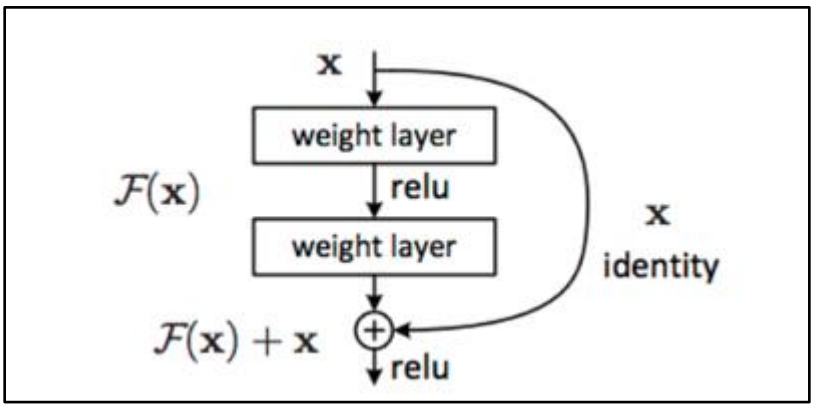


Figure 2.13: Residual learning: a building block (Residual Block) (He et al. 2015).

Figure 2.13 above displays the residual block in action. The main thing to focus here is the identity mapping, X. This identity mapping does not have any parameters and functions to add the output from the previous layer to the layer ahead. However, sometimes x and F(x) will not have the same dimension. Considering that a convolution operation shrinks the spatial resolution of an image, e.g. a 3x3 convolution on a 32 x 32 image results in a 30 x 30 image. The identity mapping is multiplied by a linear projection W (weight) to expand the channels of shortcut to match the residual (Shorten 2019). This allows for the input x and F(x) to be combined as input to the next layer.

Equation used when F(x) and x have a different dimensionality:

$$y = F(x, \{W_i\}) + W_s x$$

ResNet consists of one convolution and pooling step followed by 4 layers of the same pattern. They perform 3x3 convolution with a fixed feature map dimension [64, 128, 256, 512], bypassing the input every 2 convolutions using identity mapping. Moreover, the width (W) and height (H) dimensions remain constant during the entire layer.

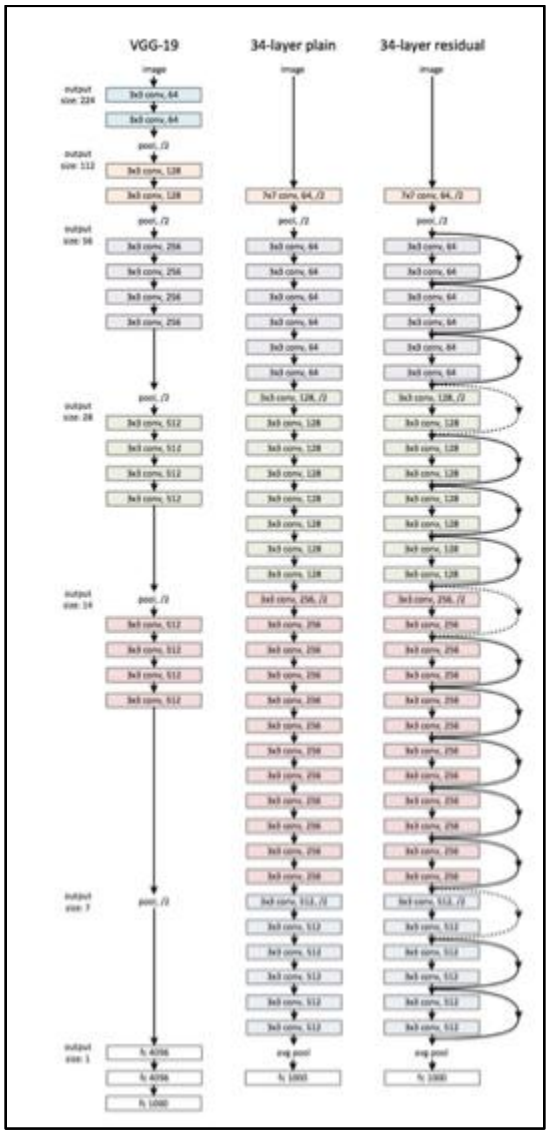


Figure 2.14: Example network architectures for ImageNet. **Left**: the VGG-19 model as a reference. **Middle**: a plain network with 34 parameter layers. **Right**: a residual network with 34 parameter layers. The dotted shortcuts increase dimensions (He et al. 2015).

According to Figure 2.14 above, the dotted line represents a change in the dimension of the input volume (reduction). This reduction between layers is achieved by an increase on the stride, from 1 to 2, at the first convolution of each layer; instead of by a pooling operation. The image below is a summary of output size at individual layers and the dimensions of the convolutional filters at every point.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	[3×3, 64; 3×3, 64] ×2	[3×3, 64; 3×3, 64] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3
conv3_x	28×28	[3×3, 128; 3×3, 128] ×2	[3×3, 128; 3×3, 128] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×8
conv4_x	14×14	[3×3, 256; 3×3, 256] ×2	[3×3, 256; 3×3, 256] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×23	[1×1, 256; 3×3, 256; 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512; 3×3, 512] ×2	[3×3, 512; 3×3, 512] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 2.15: ResNet Architectures in different layers. Building blocks are shown in brackets with the numbers of blocks stacked. Downsampling is performed by conv3_1, conv4_1, and conv5_1 with a stride of 2 (He et al. 2015).

2.5 Object Detection CNN Architectures

In the last few years (since CNN) computer vision has gained enormous amounts of traction and self-driving cars have been the center of attention. Object detection is also an integral part of computer vision. Object detection helps for estimation of poses, vehicle tracking, control, etc. The difference between algorithms of object detection and classification is that with algorithms of detection, a bounding box is drawn around the object of interest in an image to locate it. Also, there can be many bounding boxes representing different objects of interest inside the image. There is not just one bounding box drawn in an object detection case.

The major reason a standard CNN cannot proceed with this problem is that, the length of the output layer is variable— not constant, this is because the number of occurrences of the objects of interest is not fixed. A naive way of solving this problem would be to use a CNN to identify the object in different areas of interest in the image. The problem with this approach is that the objects of interest may have spatial locations and different aspect ratios within the picture. Therefore, a large number of regions are required and this could blow up computationally. Thus, algorithms such as the CNN (R-CNN) Region (Girshick et al., 2014), YOLO (Redmon et al., 2016) etc. have been developed to locate and find these occurrences rapidly.

2.5.1 R-CNN

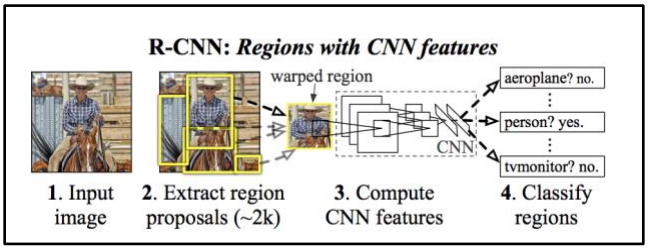


Figure 2.16: R-CNN architecture (Girshick et al., 2014).

Girshick et al. (2014) suggested a method to overcome the problem of choosing a large number of regions by selecting only two thousand regions in the image, and were termed regional proposals. These 2000 region proposals are generated using this selective search algorithm:

1. Generate initial sub-segmentation by generating many candidate regions.
2. Use greedy algorithm to recursively combine similar regions into larger ones.
3. Use the generated regions to produce the final candidate region proposals.

These 2000 candidate region proposals are warped into a square and fed into a CNN that produces a 4096-dimensional feature vector as output. The CNN acts as a feature extractor and the output dense layer consists of the features extracted from the image and the extracted features are fed into a Support Vector Machine (SVM) to classify the presence of the object within that candidate region proposal. While predicting the existence of an object within the region proposals, the algorithm also predicts four offset values to increase the accuracy of the bounding box. For example, given a region proposal, the algorithm would have predicted the presence of a person but the face of that person within that region proposal could've been cut in half. Therefore, the offset values help in adjusting the bounding box of the region proposal.

Problems with R-CNN:

- It still takes a huge amount of time to train the network as the neural network would have to classify 2000 region proposals per image.
- It cannot be implemented in real time as it takes around 47 seconds for each test image.
- The selective search algorithm is a fixed algorithm. Therefore, no learning is happening at that stage. This could lead to the generation of bad candidate region proposals.

2.5.2 Fast R-CNN

The same team (Girshick et al., 2015) therefore overcome some drawbacks of R-CNN to build a faster object detection algorithm called Fast R-CNN. Nevertheless, the CNN input image is fed in to create a convolutional feature map instead of feeding the regional proposals. From the convolutional feature map, the region of proposals is identified and wrapped into squares, and by using a RoI pooling layer, the region of proposals are reshaped into a fixed size so that it can be fed into a fully connected layer. From the RoI feature vector, a softmax layer is used to predict the class of the proposed region and also the offset values for the bounding box.

The reason why ' Fast R-CNN ' is faster than R-CNN is that the region proposals of 2000 do not always need to be added to the CNN. Rather, the convolution process is only performed once per image and a feature map is generated from it.

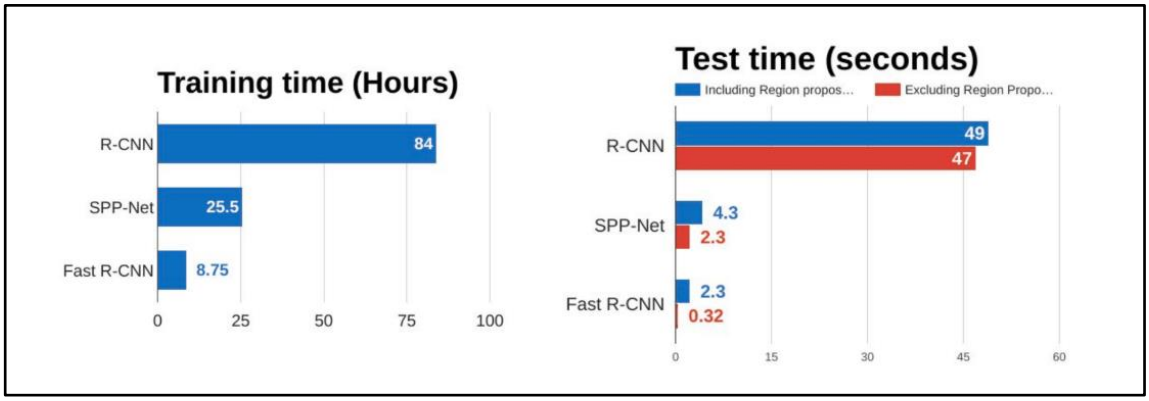


Figure 2.17: Fast R-CNN is significantly faster in training and testing sessions over R-CNN. However, the performance of Fast R-CNN when using region proposals slows down the algorithm significantly when compared to not using region proposals. Therefore, region proposals become bottlenecks in Fast R-CNN algorithm affecting its performance (John et al., 2019).

2.5.3 YOLO - You Only Look Once

All of the previous object detection algorithms are called double-shot detectors because they use regions to localize the object within an image, hence, the network does not actually 'look' at the complete image but only parts of the image which have high probabilities of containing the object. A single-shot detector like YOLO or You Only Look Once (Redmon et al., 2016) is an object detection algorithm much different from the region based algorithms seen above. In YOLO a single CNN predicts the bounding boxes and the class probabilities for these boxes.

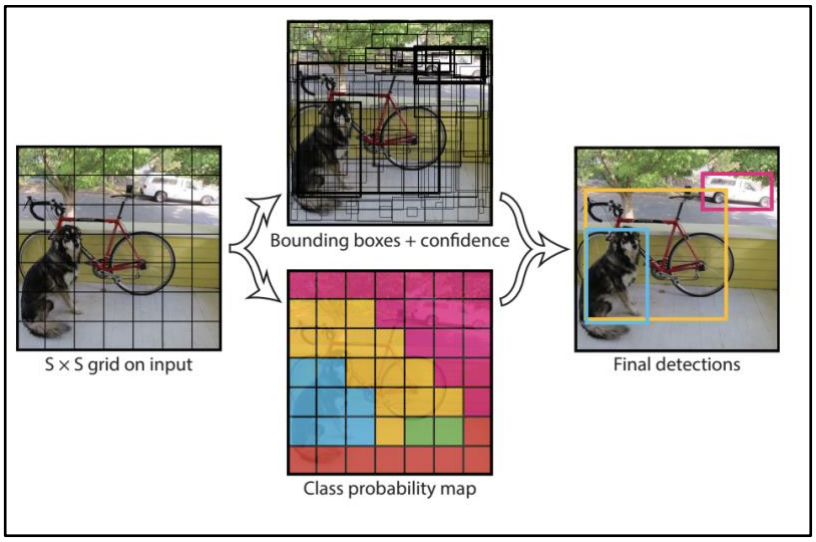


Figure 2.18: YOLO models detection as a regression problem. It divides the image into an $S \times S$ grid and for each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities. These predictions are encoded as an $S \times S \times (B * 5 + C)$ tensor (Redmon et al., 2016).

YOLO works by taking an image and splitting it into an SxS grid, within each of the grid it takes m bounding boxes. For each of the bounding box, the network outputs a class probability and offset values for the

bounding box. The bounding boxes having the class probability above a threshold value is selected and used to locate the object within the image.

YOLO is orders of magnitude faster (45 frames per second) than other object detection algorithms. The limitation of YOLO algorithm is that it struggles with small objects within the image, for example it might have difficulties in detecting a flock of birds. This is due to the spatial constraints of the algorithm.

2.6 Conclusion

From this chapter, it is found that patterns can be a set of measurements or observations, usually represented in vector notation, whereas features are the measurements extracted from the patterns. And for pattern recognition, there are statistical and structural approaches. On the other hand, neural networks are a set of algorithms, modeled loosely after the human brain, that are designed to recognize patterns and CNNs are a variance of neural networks, hierarchical neural network for recognition, particularly images. Besides, being motivated by the ImageNet Large Scale Visual Recognition Challenge, researchers have created many CNN architectures to compete against other architectures and make breakthroughs and improvements. Finally, based on the CNN architectures, R-CNNs are created and designed for object detection.

Chapter 3

Research Methodology

3 Research Methodology

In this chapter, the methodology of this research shall be revealed. Topics such as the proposed framework, dataset and evaluation method shall be discussed. These subsets of the methodology lay in the scope of practicality and real world implementations that is also backed by their theories.

3.1 Proposed Framework

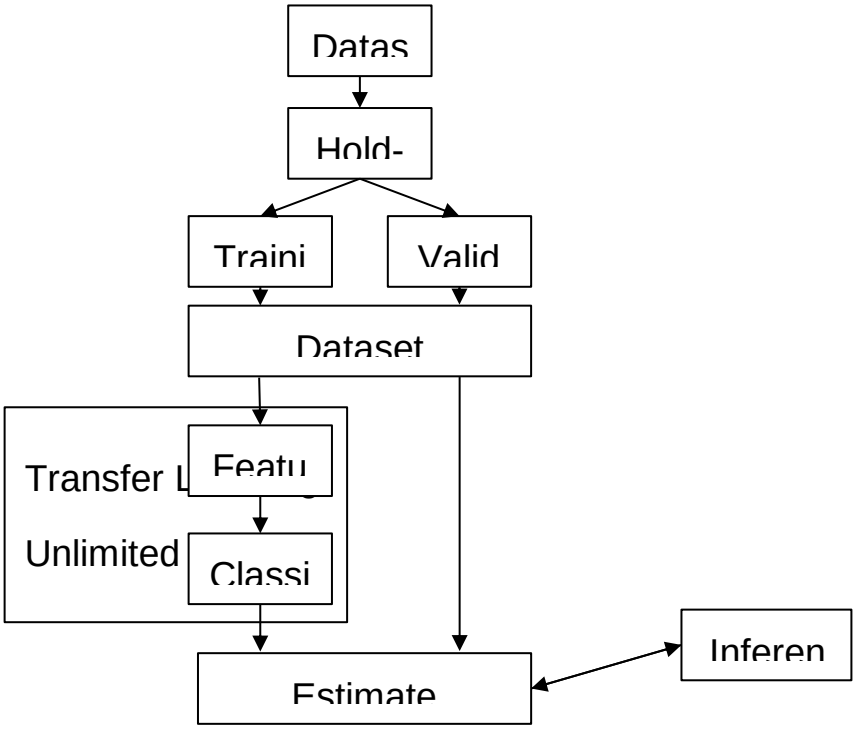


Figure 3.1: Flow of proposed framework.

Figure 3.1 illustrates a straightforward flow diagram of the proposed framework which can be summarized as follows:

1. Dataset:
   DeepFashion (Liu et al., 2016) is chosen as the dataset. It is acquired from the Multimedia Laboratory of Chinese University of Hong Kong. It consists of a total number of 800,000 annotated images, but for clothing image retrieval, there are only 300,000 pictures available. This dataset can only be used for non-commercial research purposes.

2. Dataset Splitting:
   The holdout method during the evaluation phase requires the dataset to be split into a training set, and a validation set. The DeepFashion team has prepared these two sets, which the format consists of 250,000 pictures, and the latter has 50,000 pictures.

3. Dataset Conversion:

FashionNet (Liu et al., 2016) - a VGG-16 based architecture used for the original DeepFashion paper uses a non-standard format of annotation and labels for all images, which are also stored in multiple text files. By converting this dataset to the TFRecord format that is supported by TensorFlow (Abadi et al., 2016), a greater degree of flexibility is obtained as most functions offered by TensorFlow is now available. TFRecord also improves learning performance, as well as a slight gain in accuracy of the final model.

4. Transfer Learning and Evaluation:
   A FasterRCNN-Inception-v2 model pre-trained on the COCO (Lin et al., 2014) dataset is used for transfer learning. This phase is split into two parts - feature extraction and classification. Faster R-CNN (Szegedy et al., 2015) hosts extra layers on the Inception-v2 (Szegedy et al., 2016) base model, which hugely improves object detection. The training set has unlimited access to the neural network (feature layers and classifiers). Whereas the validation set only has one shot access to estimate the accuracy of the network, which is part of the evaluation phase. TensorFlow Object Detection API (Huang et al., 2017) is used as the base framework for all the tasks mentioned here, whereas TensorBoard is used to visualize real-time results.

5. Inferencing and Testing:
   Inferencing and testing on both still image and motion image files are provided built-in by the TensorFlow Object Detection API. If a live video feed is required, e.g. external camera or webcam, OpenCV (Bradski, 2000) can be used.

A large magnitude of this proposed framework is common among many image classification tasks, especially the part about transfer learning. The hold-out method is also commonly used to avoid overfitting. However, dataset conversion is not typically performed; yet the increased performance that it potentially provide is found to be beneficial, more of this is explained in Section 3.1.1.

The flexibility and performance TensorFlow provides is optimal for the needs of this research when choosing between many others such as pytorch (Paszke et al., 2017), and Keras (Chollet, 2015). TensorFlow 2.0 is still in beta stage and is lacking for several sub-frameworks such as Faster R-CNN (Ren et al., 2017) (TensorFlow, 2019), as such, TensorFlow 2.0 is not considered. Details about TensorFlow and Faster R-CNN is revealed in Section 3.1.2.

3.1.1 Dataset Conversion

DeepFashion (Liu et al., 2016) is a rich dataset which can fulfill all the needs for image classification tasks. However, it is also made and catered for image classification framework of that time, namely FashionNet (Liu et al., 2016) which is based on VGG-16 pretrained model.

The annotation data in DeepFashion is rather unusually arranged and can be rather inefficient in terms of preprocessing work needed before used for training. For instance, all annotation data in DeepFashion, as

well as labels of training, validation, and testing image, are all written in multiple text files. This in turn requires extensive time to implement other frameworks not designed for the dataset.

As such, converting the dataset to TFRecord format can standardize this dataset for multiple use cases based on the TensorFlow framework. The TFRecord format can also help improve performance as it can store thousands or millions of files into one single file or set of files in serialized form. The serialization is especially important when the dataset is streamed over a network for cross-domain training and processing. There are guidelines on the TensorFlow tutorials on how to convert datasets to TFRecord format.

For this research, the dataset has been converted from hundreds of thousands of image files to three TFRecord formatted files, namely for training, validation, and testing, respectively.

3.1.2 Faster R-CNN on TensorFlow

Faster R-CNN is a model built for efficient object detection, which includes multiple classification and localization outputs. It adds extra layers on a base model such as Inception-v2 to improve performance for object detection. It was originally built for Caffe (Jia et al., 2014) but alternate implementations exist. One of the many alternate implementations of Faster R-CNN is the Object Detection API of TensorFlow framework, though many other choices of object instance segmentation frameworks are also available. Historically, Faster R-CNN is built to provide better performance compared to Fast R-CNN (Girshick, 2015).

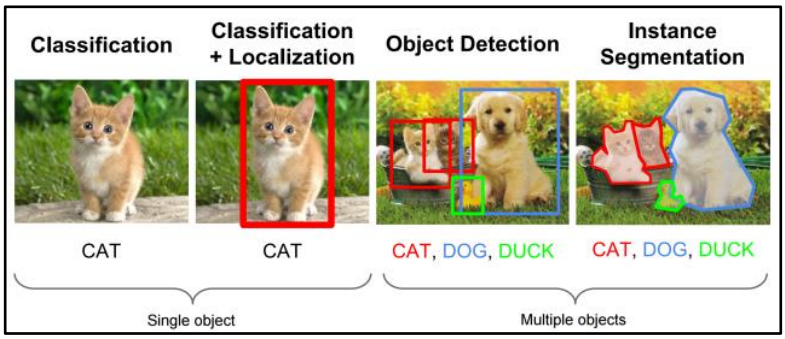


Figure 3.5: Differences between different image classification output method (Quaknine, 2018).

3.1.3 Considerations Against Mask R-CNN

Mask R-CNN (He et al., 2017) is a state of the art model built for object instance segmentation tasks and a new masking capability in replacement to bounding boxes. Mask annotation is a prerequisite in using this framework, which DeepFashion lacks. It is therefore not considered for this study.

3.1.4 Transfer Learning on FasterRCNN-Inception-v2 using TensorFlow

TensorFlow is a robust solution for many machine learning tasks, including transfer learning. For this research. FasterRCNN-Inception-v2 pre-trained model is chosen as a base model for its efficiency and excellent accuracy. This model is pre-trained on the COCO (Lin et al., 2014) dataset which is a large dataset consists of 328,000 images. Configurations and hyper-parameters catered for this model is provided by the object detection API of TensorFlow, TensorFlow allows instances of training and evaluation steps to be performed automatically.

On fine-tuning for transfer learning, since DeepFashion is a large dataset but its features to extract are different from the original COCO dataset, only the architecture and the weights of the FasterRCNN-Inception-v2 model is used for transfer learning, hence, its previous knowledge fron the COCO dataset is discarded (Marcelino, 2018).

TensorBoard is used to illustrate the learning process in real time. The learning instances is set to run indefinitely until manually terminated. Criteria of termination is based on the scalar accuracy figures provided by TensorBoard. Those figures are observed to determine the state of the learning process; whether its accuracy/error has reached a global maxima/minima, whether it has converged, or even when overfitting starts to occur.

3.1.5 Inferencing and Testing

Object detection API from TensorFlow is catered for motion image inferencing. FasterRCNN-Inception-V2 is measured to be able to run each frame at about 58ms, which is about 17 frames per second on the COCO dataset. Live feed can also be implemented through OpenCV, with simultaneous processing over the live feed data and real time output with object detection boxes and labels. However, the API can also be used for simple still image inferencing.

Inference graphs are also generated periodically during the learning phase, thus inferencing is also performed simultaneously with learning for data generation on TensorBoard. With the right configuration, image or video inferencing can also be performed periodically during the learning phase.

Model name	Speed (ms)	COCO mAP[^1]	Outputs
ssd_mobilenet_v1_coco	30	21	Boxes
ssd_mobilenet_v1_0.75_depth_coco ☆	26	18	Boxes
ssd_mobilenet_v1_quantized_coco ☆	29	18	Boxes
ssd_mobilenet_v1_0.75_depth_quantized_coco ☆	29	16	Boxes
ssd_mobilenet_v1_ppn_coco ☆	26	20	Boxes
ssd_mobilenet_v1_fpn_coco ☆	56	32	Boxes
ssd_resnet_50_fpn_coco ☆	76	35	Boxes
ssd_mobilenet_v2_coco	31	22	Boxes
ssd_mobilenet_v2_quantized_coco	29	22	Boxes
ssdlite_mobilenet_v2_coco	27	22	Boxes
ssd_inception_v2_coco	42	24	Boxes
faster_rcnn_inception_v2_coco	58	28	Boxes
faster_rcnn_resnet50_coco	89	30	Boxes
faster_rcnn_resnet50_lowproposals_coco	64		Boxes
rfcn_resnet101_coco	92	30	Boxes
faster_rcnn_resnet101_coco	106	32	Boxes
faster_rcnn_resnet101_lowproposals_coco	82		Boxes
faster_rcnn_inception_resnet_v2_atrous_coco	620	37	Boxes
faster_rcnn_inception_resnet_v2_atrous_lowproposals_coco	241		Boxes
faster_rcnn_nas	1833	43	Boxes

Figure 3.6: Differences in speed of different pre-trained models used on different object detection frameworks (TensorFlow, 2019).

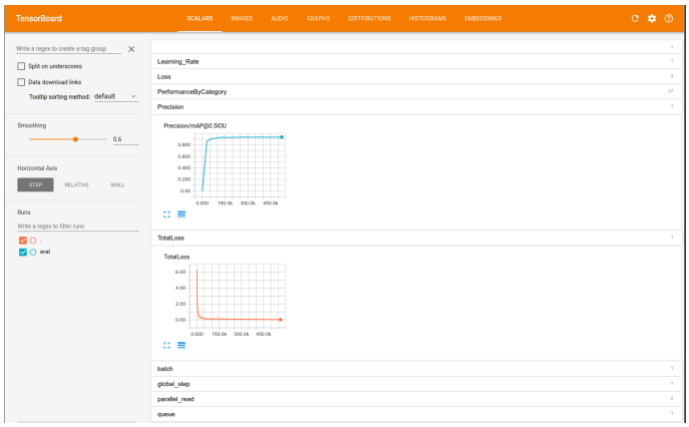


Figure 3.7: Real-time graph generation on TensorBoard (TensorFlow, 2019).

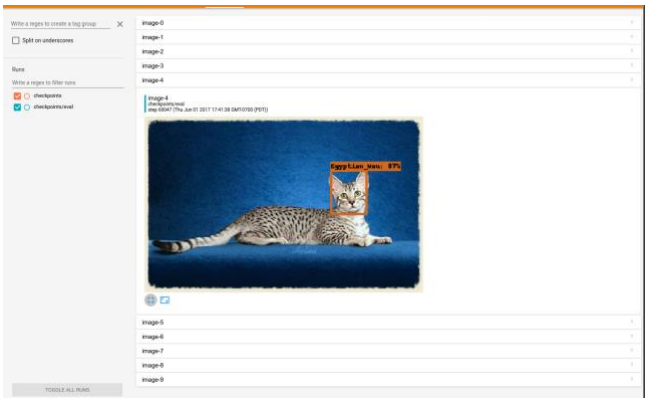


Figure 3.8: Real-time image inferencing on TensorBoard (TensorFlow, 2019).

3.2 Dataset

The adoption of clothes recognition algorithms in real world applications often faces three fundamental challenges (Kiapour et al., 2015). First, the large variations in style, texture, and cutting of clothes tend to confuse many systems. Second, deformation and occlusion are common among clothing items. Third, serious variations are often exhibited in clothing images when they are taken under different scenarios, e.g. a selfie photo versus an online shopping photo of the same product.

Attempts by many studies to handle the above challenges focus on the annotation of the clothes datasets either with semantic attributes (Chen et al., 2012), clothing locations (Luo et al., 2013), or cross-domain image correspondences (similarities between shop images and street images) (Huang et al., 2015). However, none of the aforementioned datasets contain multiple types of annotation.

3.2.1 DeepFashion

DeepFashion (Liu et al., 2016) contains these multiple types of annotation and the research team has claimed that clothes recognition can be benefited from learning these annotations jointly. The ultimate description of DeepFashion as given by the researchers was "a comprehensively annotated clothes dataset that contains massive attributes, clothing landmarks, as well as cross-pose/cross-domain correspondences of clothing pairs".

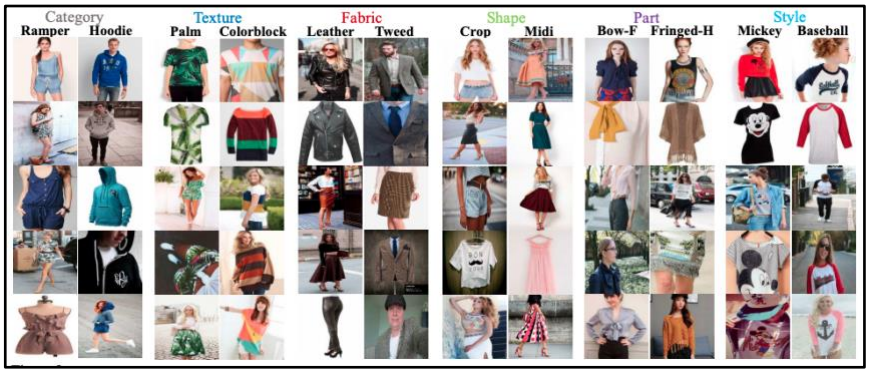


Figure 3.9: DeepFashion features 50 categories and 1,000 attributes for attonations of clothing images; the attributes consists of five groups: texture, fabric, shape, part, and style (Liu et al., 2016).

DeepFashion has several advantages over most other datasets:

1. Comprehensiveness - each image in DeepFashion is annotated with category, attributes, landmarks, and its cross-pose/cross-domain pair. It has 50 categories and 1,000 attributes. The attributes consists of five groups: texture, fabric, shape, part, and style. The landmarks are also grouped to upper-body, lower-body and full body items.
2. Scale - DeepFashion contains over 800,000 annotated clothing images, far richer than many other datasets. This scale tends to reduce risk of overfitting.

3. Availability - DeepFashion is made public by the research team for the benefit of all research regarding fashion recognition and retrieval.

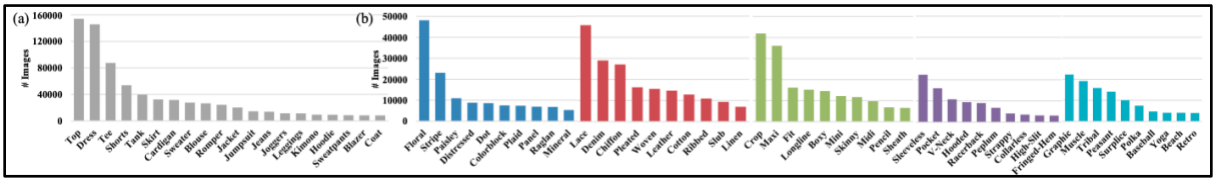


Figure 3.10: Distribution of images of the top-20 categories and top-10 attributes in each attribute group (Liu et al., 2016).

Figure 3.11: Landmarks and pair annotation in DeepFashion (Liu et al., 2016).

3.2.2 Considerations Against DeepFashion2

Figure 3.12: From (a), DeepFashion only has one item per image, annotated with 4-8 sparse landmarks. The bounding boxes are estimated from the landmarks, which cause them to be noisy. In (b), each image can have one to seven items. Each item is manually labeled with bounding box, mask, and dense landmarks (20 average per item) (Ge et al., 2019).

DeepFashion2 (Ge et al., 2019) is published recently to resolve several weaknesses of the original DeepFashion (Liu et al., 2016), among them are:

1. Single item per image
2. Sparse landmarks (4-8 only)
3. No per-pixel masks

Though DeepFashion2 is even more comprehensive than DeepFashion in terms of its annotation data, it is ultimately not chosen in this research due to its limited number of clothing categories compared to its predecessor (13 versus 50). The number of category labels is important for the evaluation work of formality of clothing, as more category labels would be able to provide a more robust formality rating.

3.3 Evaluation Method

Two prominent evaluation methods are holdout method and cross-validation (Kohavi, 2001). The holdout method can be used as a large dataset is chosen for this research. More about this method is discussed in Section 3.3.1.

In Section 3.3.2, method to evaluate classificated images is revealed.

3.3.1 Holdout

The holdout method, sometimes called test sample estimation, partitions the data into two mutually exclusive subsets called a training set and a test set, or holdout set (Kohavi, 2001). It is common to designate 2/3 of the data as the training set and the remaining 1/3 as the validation set. The training set is given to the inducer, an induction algorithm, and the induced classifier is tested on the validation set. Formally, let D_h, the holdout set, be a subset of D of size h, and let D_t be D \ D_h.

The holdout estimated accuracy is defined in this formula:

$$acc_h = \frac{1}{h} \sum_{(v_i, Y_i) \in D_h} \delta(L(D_t, v_i), Y_i), where\ \delta(i;j) = 1\ if\ \ i = j\ and\ 0\ otherwise$$

Assuming that the inducer's accuracy increases as more instances are seen, the holdout method is a pessimistic estimator because only a portion of the data is given to the inducer for training. The more instances is left for the validation set, the higher the bias of estimate; however, fewer validation set instances means that the confidence interval for the accuracy will be wider. The training set is also advised to be shuffled to avoid overfitting.

To translate these theories in practical usage, the dataset is split into training set and validation set, in case for DeepFashion 1/6 of the dataset is already pre-partitioned as validation set. The shuffling of the dataset is performed when the dataset is being converted into the TFRecord format, thus able to reduce overfitting.

3.3.2 Evaluation of Classification

The main purpose of performing evaluation is to compare the accuracy of the training set against data that is not present in that set (Skalski, 2018). The mere output of the training set is mostly biased and overfitted.

As discussed in Section 3.3.1, the dataset is split into two parts. The validation set is an excellent set of data to reduce the biases of the classification.

To obtain more data is the most preferred way to prevent overfitting. Fortunately, DeepFashion is rich enough to reduce this risk. However, there are still risks that a dataset might caused the neural network to have high variance. Ergo, the tensor is configured to apply regularization. Regularization involves adding an extra element to the loss function, which punishes the model for being too complex or, in simple words, for using too high values in the weight matrix. This way, its flexibility is limited, but is also encouraged to build solutions based on multiple features. Two popular versions of this method are L1 - Least Absolute Deviations (LAD) and L2 - Least Square Errors (LS).

Equations that describe L1 and L2 regularizers:

$$J_{L1}(W,b) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{i}, y^{i}) + \lambda \left\| W \right\|_1, \qquad \left\| W \right\|_1 = \sum_{j=1}^{n_x} |W_j|$$

$$J_{L2}(W,b) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{i}, y^{i}) + \lambda \left\| W \right\|_2, \qquad \left\| W \right\|_2 = \sum_{j=1}^{n_x} W_j{}^2$$

In practical, during the evaluation phase, the L1 regularizer is used because it reduces the weight values of less important features to zero, very often eliminating them completely from the calculations. In a way, it is a built-in mechanism for automatic feature selection. Moreover, L2 does not perform very well on datasets with a large number of outliers and large number of features. The DeepFashion dataset contains many annotations and street domain images, thus, the L2 regularizer is likely to perform poorly.

3.4 Summary

There exists many methods in various levels of the process of image classification and pattern recognition that aims to achieve optimal results within known use cases. The generic levels of the process would be: First, the methods in obtaining the input dataset for the image classification tasks - either still images or motion images. Second, the methods used for image classification such as the framework used - TensorFlow (Abadi et al., 2016), pytorch (Paszke et al., 2017), Keras (Chollet, 2015), etc. Third, whether a pre-trained model is used, if so, which pre-trained model - Resnet-50 (He et al., 2015), VGG-16 (Simonyan and Zisserman, 2015), Inception-v2 (Szegedy et al., 2016), etc. Forth, the framework or tools used to output the results - bounding box, masking, labelling, or plain text output. And finally, the methods and approaches in evaluating the results - whether the holdout method, or the cross-validation method, and to choose between the L1 and L2 regularizers.

All of these choices of methods, combined with careful configuration, makes up a framework on neural network that is able to serve required needs.

Chapter 4

Theory Background

4 Theory Background

Theoretical research upon Inception-v1 and Inception-v2 - the chosen model, and transfer learning is presented in this chapter. For the Inception model architectures, 1x1 convolutions, global average pooling, and the Inception module is explained. Differences between Inception-v1 and Inception-v2 are also detailed in this chapter.

4.1 Inception-v1

Inception-v1 a.k.a. GoogLeNet (Szegedy et al. 2014) was the winner of the ILSVRC 2014 competition, achieving a top-5 error rate of 6.67%. The architecture was built upon an existing network, LeNet-5 (Lecun et al. 1998), thus "LeNet" is contained within the name. It consisted of a 22-layer deep CNN but reduced the number of parameters from 60 million of AlexNet (Krizhevsky et al. 2012) to 4 million.

Some issues the Faster R-CNN is aimed to address:

- Important parts in the image can have extremely large variation in size. For instance, a set of images with a dog, the area occupied by the dog is different in each image.
- Because of this huge variation in the location of the information, choosing the right kernel size for the convolution operation becomes tough. A larger kernel is preferred for information that is distributed more globally, and a smaller kernel is preferred for information that is distributed more locally.
- Very deep networks like VGGNet (Simonyan and Zisserman 2015) are prone to overfitting. It also hard to pass gradient updates through the entire network.
- Naively stacking large convolution operations is computationally expensive.

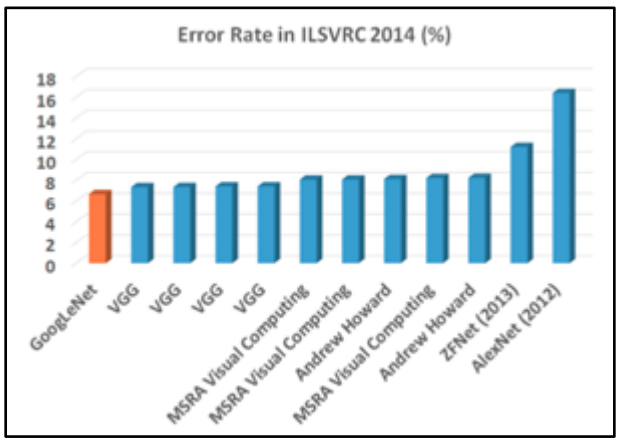


Figure 4.1: ILSVRC 2014 Error Rate (%). Inception-v1 (GoogLeNet) having the lowest rate at 6.67%, followed by VGG at 7.32% (Tsang, 2018).

Inception-v1 architecture contains various new techniques such as 1 x 1 Convolutions, global average pooling and the implementation of a new module called Inception module. This new module works by stacking multiple outputs of the same input through different sizes/types of convolutions (Tsang 2018).

1) The 1x1 Convolutions

- Originally introduced by Network in Network (Lin et al. 2013).
- Used as a dimension reduction module to reduce the computation. By reducing the computation bottleneck, depth and width can be increased.

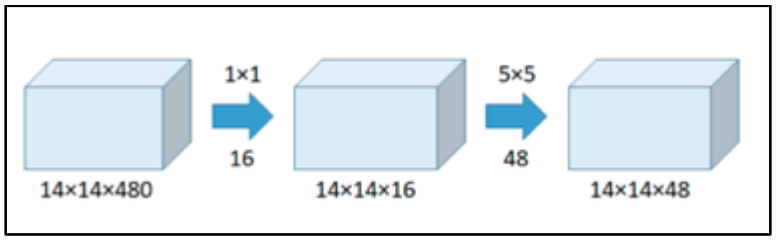


Figure 4.2: Example of 1x1 convolutions followed by a 5x5 convolution using a 14x14x480 input (Tsang 2018).

Through the use of Figure 4.2 above, the total number of operations needed is computed by:

Number of operations for 1×1 = (14×14×16) × (1×1×480) = 1.5M

Number of operations for 5×5 = (14×14×48) × (5×5×16) = 3.8M

Total number of operations = 1.5M + 3.8M = 5.3M

As opposed to performing 5x5 convolutions without the use of 1x1 convolution:

Number of operations = (14×14×48) × (5×5×480) = 112.9M (Compared to 5.3M from added a 1x1 convolution)

In conclusion, using a 1x1 convolutions allows the inception module to be built without increasing the number of operations, which in turn reduces model size.

2) Global average pooling

- In Inception-v1 (GoogLeNet), global average pooling is used almost at the end of network by averaging each feature map from 7×7 to 1×1
- Previously, fully connected (FC) layers are used at the end of network, such as in AlexNet. All inputs are connected to each output.
- The authors found that a move from fully connected layers to average pooling improved the top-1 accuracy by about 0.6%.

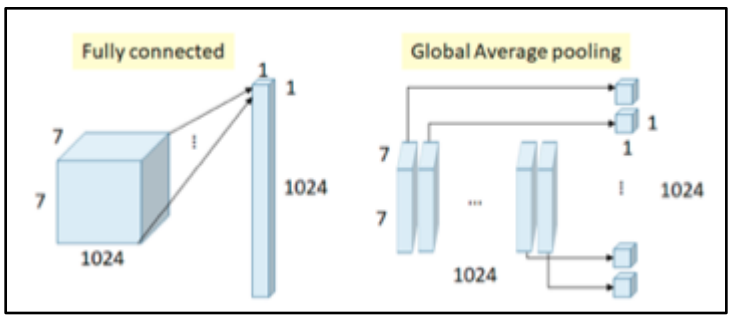


Figure 4.3: Fully Connected Layer VS Global Average Pooling (Tsang 2018).

3) Inception Module

- The Inception module works through the use of parallel paths with different receptive field sizes and operations are meant to capture sparse patterns of correlations in the stack of feature maps.
- Uses 1x1 convolutions for dimensionality reduction before expensive convolutions.

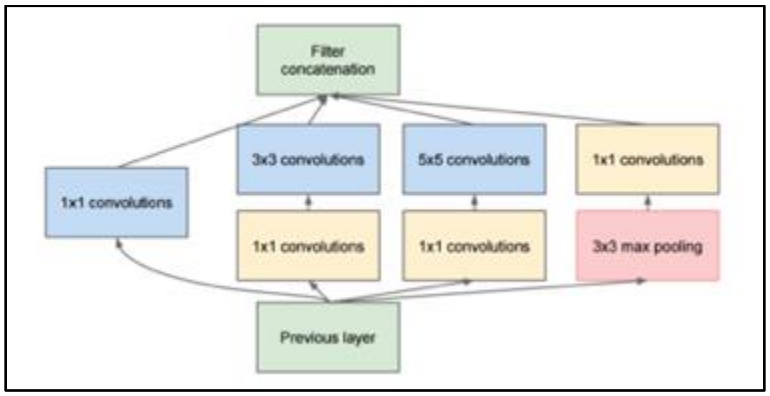


Figure 4.4: : Inception module with dimensionality reduction (1x1 convolution added) (Tsang 2018).

The network is designed in order for the network to work in individual devices, even those with limited computational resources or low memory footprint. Therefore, computational efficiency and practicality were taken into consideration. The network is 22 layers deep when counting only layers with parameters (or 27 layers including pooling). The overall number of layers (independent building blocks) used for the construction of the network is about 100 (Szegedy et al. 2014).

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Figure 4.5: Inception-v1 (GoogLeNet) incarnation of the Faster R-CNN (Szegedy et al. 2014).

4.2 Inception-v2

Inception-v2 (Szegedy et al. 2016), the chosen model for this research, is proposed by the same team from Inception-v1 the following year with a number of upgrades which increased the accuracy and reduced the computational complexity, issues that were addressed to solve were:

- Reduce representational bottleneck. The intuition was that, when convolutions did not change the dimensions of the input, neural networks tend to perform better. When there is too much loss of information, it is called "representation bottleneck".
- Convolutions can be made more efficient in computational complexity using smart factorization techniques.

A few solutions were proposed, they are:

- Factorize 5x5 convolution to two 3x3 convolution operations to improve computational speed. Although this may seem counterintuitive, a 5x5 convolution is 2.78 times more expensive than a 3x3 convolution. So stacking two 3x3 convolutions in fact leads to a boost in performance. This is illustrated in the below image.

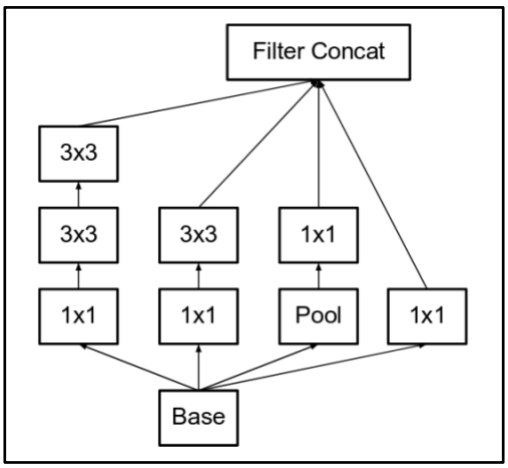


Figure 4.6: The left-most 5x5 convolution of Inception-v1 (refer Figure 4.4) is now represented as two 3x3 convolutions (Szegedy et al. 2016).

- Convolutions of filter size nxn are factorized to a combination of 1xn and nx1 convolutions. For example, a 3x3 convolution is equivalent to first performing a 1x3 convolution, and then performing a 3x1 convolution on its output. They found this method to be 33% cheaper than the single 3x3 convolution. This is illustrated in Figure 4.7:

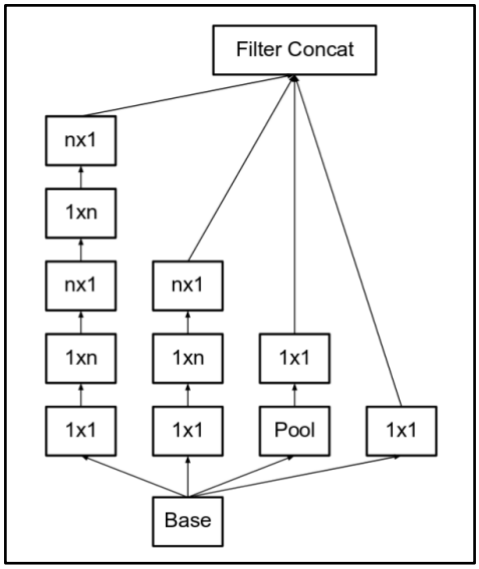


Figure 4.7: Here, put n=3 to obtain the equivalent of the Figure 4.6. The left-most 5x5 convolution can be represented as two 3x3 convolutions, which inturn are represented as 1x3 and 3x1 in series (Szegedy et al. 2016).

- The filter banks in the module were expanded (made wider instead of deeper) to remove the representational bottleneck. If the module was made deeper instead, there would be excessive reduction in dimensions, and hence loss of information. This is illustrated in Figure 4.8:

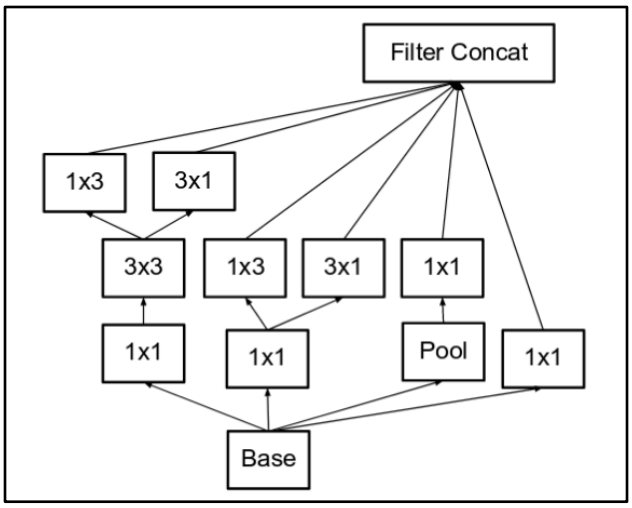


Figure 4.8: Making the inception module wider. This type is equivalent to the module shown in Figure 4.7 (Szegedy et al. 2016).

- The above three principles were used to build three different types of inception modules. The architecture is shown in Table 4.1:

type	**patch size/stride** or remarks	**input size**
conv	$3\times3/2$	$299\times299\times3$
conv	$3\times3/1$	$149\times149\times32$
conv padded	$3\times3/1$	$147\times147\times32$
pool	$3\times3/2$	$147\times147\times64$
conv	$3\times3/1$	$73\times73\times64$
conv	$3\times3/2$	$71\times71\times80$
conv	$3\times3/1$	$35\times35\times192$
$3\times$Inception	As in figure 5	$35\times35\times288$
$5\times$Inception	As in figure 6	$17\times17\times768$
$2\times$Inception	As in figure 7	$8\times8\times1280$
pool	8×8	$8 \times 8 \times 2048$
linear	logits	$1 \times 1 \times 2048$
softmax	classifier	$1 \times 1 \times 1000$

Table 4.1: Here, “figure 5” is Figure 4.6, “figure 6” is Figure 4.7, and “figure 7” is Figure 4.8 (Szegedy et al. 2016).

4.3 Faster R-CNN

In order to identify the region's proposals, R-CNN (Girshick et al. 2014) and Fast R-CNN (Girshick et al. 2015) uses selective search, which is a long and slow process that has an effect on the network's performance. Therefore, Ren et al. (2017) came up with an object detection algorithm that eliminates the selective search algorithm and lets the network learn the region proposals.

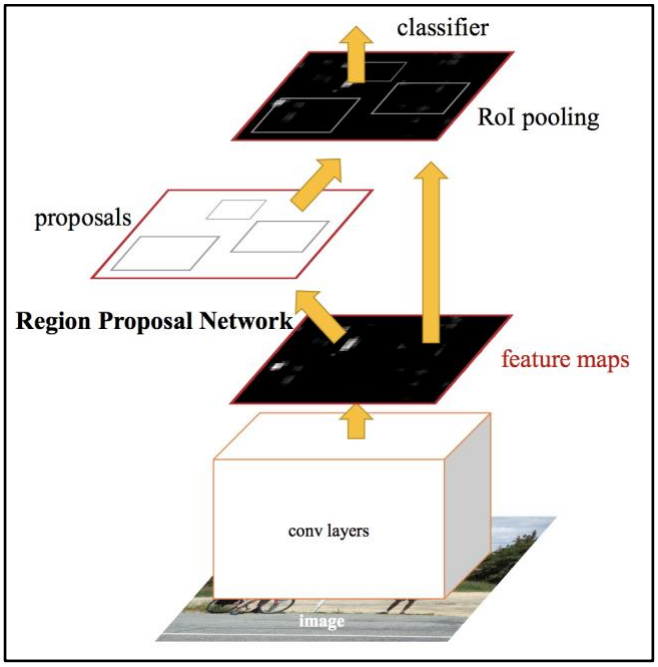


Figure 4.9: Faster R-CNN is a single, unified network for object detection. The RPN module serves as the 'attention' of this unified network (Ren et al., 2017).

Like Fast R-CNN, the image is provided as an input for a CNN that provides a convolutional feature map. However, from here, the regional proposals are predicted by a separate network, instead of using a selective search algorithm on the feature map. The predicted region proposals are then reshaped using a RoI pooling layer which is then used to classify the image within the proposed region and predict the offset values for the bounding boxes.

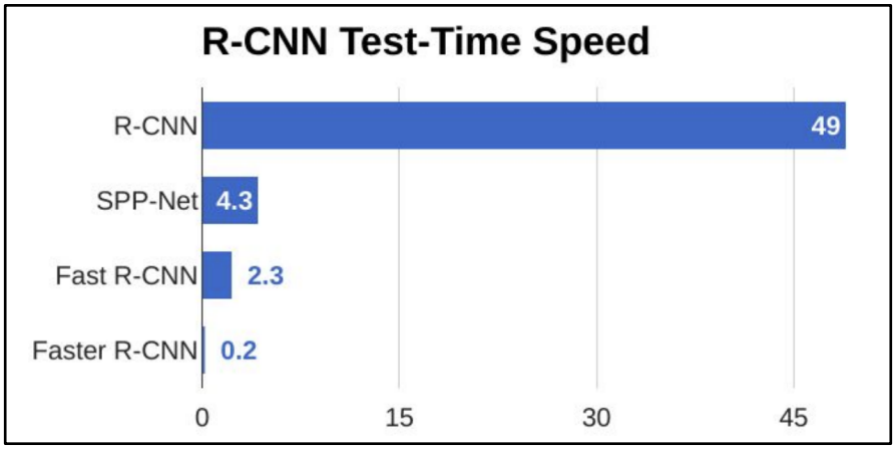


Figure 4.10: Faster R-CNN is much faster than its predecessors. To an extent that it can be used for real-time object detection (John et al., 2019).

4.4 Transfer Learning

Transfer learning is usually carried out on CNNs because they can construct accurate models in a time-consuming manner (Rawat and Wang 2017). With transfer learning, researchers can use patterns that were learned when solving a different problem on the new problem instead of learning from scratch. Previous learnings can hence be leveraged and ultimately, save a lot of time.

Transfer learning is usually expressed through the use of pre-trained models - or in particular, to repurpose them for specific uses (Chollet 2018). A pre-trained model is a model that has been trained on a large dataset to solve a problem. With transfer learning, it is common practice to import and use models from published literature due to the high calculational cost to replicate the performance of models such as ResNet, VGGNet, Mobilenet, and more.

The original classifier is removed to reorient a pre-trained model, then, a new classifier is added, which corresponds to new purposes, then the model is fine tuned to one of three policies - train the entire model, train some layers and leave the other frozen, or freeze the convolutional base.

To justify for training the entire model, usually, a large dataset and a lot of computing power is present. When all is well, the pre-trained model architecture is used and trained to match the new dataset.

However, when choosing to train some layers and leave the others frozen, the dataset is usually small and has a large number of parameters, and more layers are usually left frozen to avoid overfitting. The lower layers refer to general features (problem independent), while higher layers refer to specific features (problem dependent). In contrast, if the dataset is large and the number of parameters is small, the model can be improved by training more layers to the new task since overfitting is not an issue.

Moreover, when choosing to freeze the convolutional base, there is usually a shortage of computational power, the dataset is small, or pre-trained model is designed to solve a similar problem. The main idea is usually keep the convolutional base in its original form and then use its outputs to feed the classifier. It is

also usually corresponding to an extreme situation of the train/freeze trade-off. When doing so, the pre-trained model is used as a fixed feature extraction mechanism, which can be useful given the situation described above.

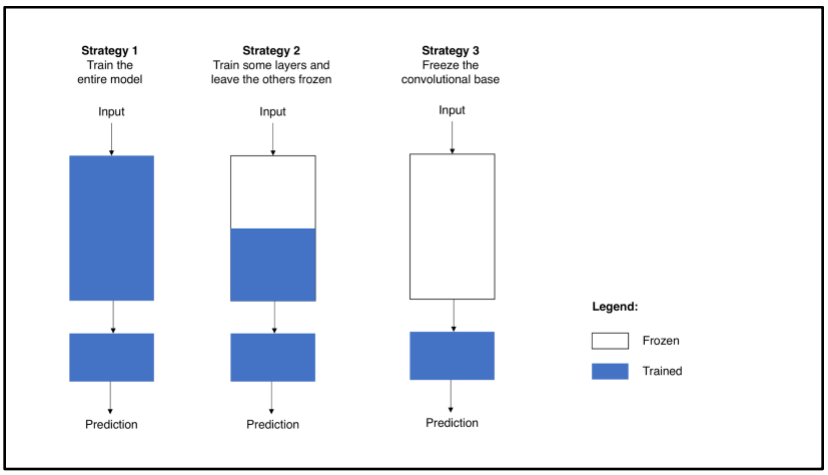


Figure 4.11: Schematics of three fine-tuning strategies (Marcelino, 2018).

Unlike Strategy 3, which application is straightforward, Strategy 1 and Strategy 2 require more handling regarding the learning rate used in the convolutional part. When a CNN-based pre-trained model is used, a learning rate is recommended because high levels of learning rate increases the risk of losing prior knowledge. Assuming that the pre-trained model has been well trained, which is a fair assumption, keeping a small learning rate will ensure the CNN weights are not distorted too soon and too much.

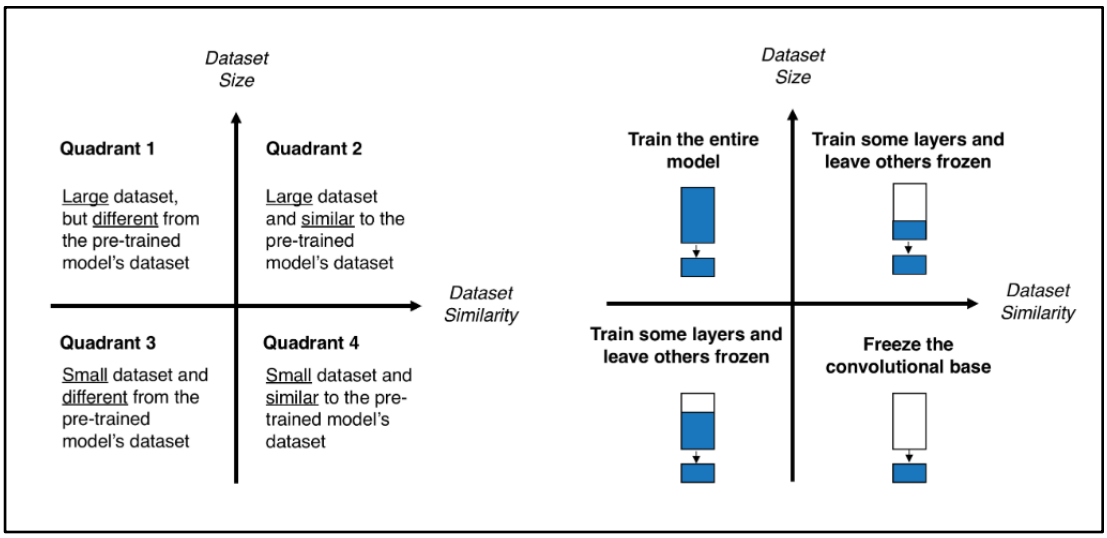


Figure 4.12: Size-Similarity matrix (left) and decision map for fine-tuning pre-trained models (right) (Marcelino, 2018).

4.4 Summary

From this chapter, it is learned that Inception-v1 is an efficient architecture with great accuracy and performance, but Inception-v2 improves upon it to be even better. The researchers (Szegedy et al., 2014) (Szegedy et al., 2016) did it by having 1 x 1 convolutions, global average pooling and the implementation of a new module called Inception module. Inception-v2 made many improvements on the Inception module by optimizing the filter layer and sizes. On the other hand, with transfer learning, previous learning can be leveraged and save a lot of time for learning. Furthermore, Faster R-CNN is faster than both the original R-CNN and Fast R-CNN, and is the first among the R-CNN architectures that can perform object detection in real-time. And finally, they are generally three strategies in transfer learning, each for different use cases.

Chapter 5

Experiment Results

5 Experiment Results

This chapter introduces and explains the implementation and results of the experiment in detail. Within the subsections of 5.1 (Setup and Configuration), the steps were performed chronologically. Section 5.2 introduces the evaluation metrics used that would be revealed in Section 5.3 and 5.4, the latter two being the results through evaluation or real world testing.

This paper is not intended to dwell into the details of the codes used in the experiment, however, a Github repository is publicly available for clarity and accessibility. Details about the repository is revealed in the next section.

5.1 Setup and Configuration

The use of Tensorflow's Object Detection API (Huang et al., 2017) simplifies the experiment process by omitting the need of self-written codes which are replaced by configuration files. However, the API is not exempted from a proper runtime environment which fulfills the dictated dependencies for the API to become functional.

The machine used in this experiment was installed with one unit of GeForce GTX 1080 with 8 gigabytes of GPU memory, software wise it was using Ubuntu 16.04.6 LTS.

This section explains the process of the aforementioned preparations and prerequisites needed before beginning the experiment. For the sake of clarity and accessibility, a Github repository for this experiment is publicly available at: https://github.com/tyrng/deepfashionDetection. Snippets of the Github repository is also attached at Appendix B.

5.1.1 Runtime Environment

Runtime environment for all stages of the experiment including dataset preprocessing, training, and evaluation are about the same with Object Detection API, which includes namely Python 3, Tensorflow GPU, Matplotlib, cocoapi, pandas, and Pillow 1.0, to name a few. It is also important to note that Nvidia CUDA is set up to allow Tensorflow to utilize the GTX 1080 GPU. The details of the runtime dependencies are provided in the aforementioned Github repository.

5.1.2 Dataset Preprocessing

The dataset was downloaded from the official DeepFashion's (Liu et al., 2016) website. The subset used was the Category and Attribute Prediction Benchmark set. Both full and compressed quality images were provided, but we used the latter as a minimum size of 256x256 pixels being sufficient for image classification (Krizhevsky et al., 2012), but it was chosen also due to hardware limitations.

A guideline to convert any custom dataset to TFRecord was given by the Object Detection API team in their Github repository. The images, annotations and labels of DeepFashion was extracted accordingly and serialized into three TFRecord files, namely for testing, training and evaluation (DeepFashion provided labels for these three sets with regard to holdout method). During the conversion process, the images were also shuffled to allow images with more variety of classes to be trained in the same batch as faster convergence was observed (Bengio, 2012) with such method.

An example code implementation can be found in the Github repository.

5.1.3 Training and Evaluation Configuration

A Faster R-CNN (Szegedy et al., 2015) pretrained model was downloaded from the Tensorflow Object Detection Model Zoo for transfer learning. The pretrained model in the model zoo includes a file for the configuration used by the Tensorflow's team to train on the COCO dataset (Lin et al., 2014).

For transfer learning on clothing image retrieval, the configuration file was edited for a better suit, details as follows:

1. The regularizer was switched from L2 to L1 because, as mentioned in Section 3.3.2, L2 regularizer tends to perform poorly on bigger dataset such as DeepFashion.
2. The batch size was defaulted to 1 due to limitations in Faster R-CNN architecture, where the batch size needs to be less than or equal to the number of GPU used, which in this case, is one.
3. The total number of steps was set to 3,000,000 so that we can train the data for 12 epochs (3,000,000 steps / 250,000 images = 12 epochs).
4. Learning rate was manually stepped with initial learning rate of 0.0002, and scheduled to 0.00002 on 900,000 steps onwards, 0.000002 on 1,200,000 steps onwards, and 0.0000002 on 2,000,000 steps onwards. The last schedule was added due to our total number of steps of 3,000,000 being too far off from 900,000 steps' schedule.

A copy of the configuration file used in this experiment is available in the Github repository.

The API would schedule an evaluation phase about 10 minutes after each training phase, and the training phase would restart again until the specified total number of steps is met.

For evaluation, one of every 50 evaluation images is used for evaluation, which sums up to 1,000 evaluation images or 2 percent of total evaluation images during each evaluation phase. Not all evaluation images were

used because of hardware limitations, as the evaluation phase for 1,000 images has already taken approximately 15 minutes each.

5.2 Evaluation Metrics

This section is set to introduce the evaluation metrics used in this paper to measure the performance of the object detection model. The COCO (Lin et al., 2014) evaluation metrics is mainly used as provided by the API, however, for the sake of fair comparisons as the original DeepFashion paper used top-n accuracy to evaluate their models, the top-n accuracy and confusion matrix are also included as parts of the evaluation metrics in this experiment.

5.2.1 Confusion Matrix

A classifier marks examples as either positive or negative in a binary decision problem (Davis and Goadrich, 2006). In a structure known as a confusion matrix or contingency table, the classifier's decision can be interpreted. There are four groups in the confusion matrix: True Positive (TP) are cases that are correctly classified as positive. False positives (FP) refer to negative examples incorrectly labeled as positive. True negatives (TN) refer to negatives examples correctly labeled as negative. Finally, false negatives (FN) apply to positive examples incorrectly labeled as negative.

	actual positive	actual negative
predicted positive	TP	FP
predicted negative	FN	TN

Figure 5.1: A simple confusion matrix (Davis and Goadrich, 2006).

In Figure 5.1, the confusion matrix is shown in its most basic form. TP and TN reflects accurate prediction from the actual reality, while FP (or Type 1 error) and FN (Type 2 error) reflects incorrect prediction from the actual reality.

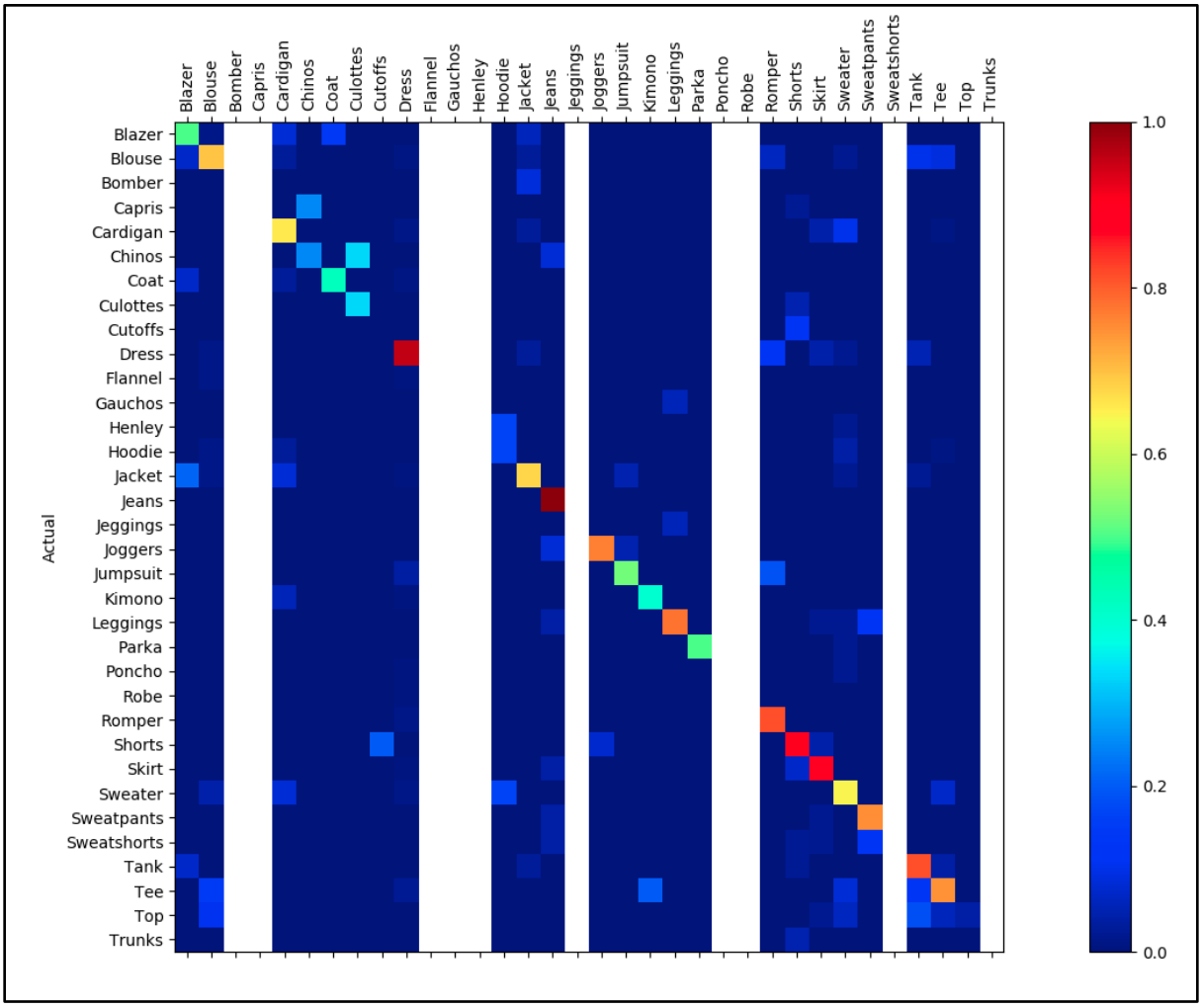


Figure 5.2: Confusion matrix from evaluation results of this experiment.

However, the confusion matrix in Figure 5.1 represents a two-dimensional problem. Figure 5.2 shows a confusion matrix from the actual experiment of this paper, which is multidimensional. Much like a two-dimensional confusion matrix, the correct decisions made by the classifier are represented on the diagonal angle of top-left to bottom right, while the rest would be incorrect decisions.

By reading from left to right to find out the distribution of the predicted results from the actual class, one can determine how ‘confused’ the classifier is when the results are not located in the diagonal.

5.2.1 Precision and Recall

Precision and recall are often confused as similar or same with accuracy. However, they both often hold deeper meanings than accuracy.

$$p = \frac{True\ Positive}{Actual\ Resuls} = \frac{TP}{TP + FP}$$

$$r = \frac{True\ Positive}{Predicted\ Results} = \frac{TP}{TP + FN}$$

As the sum of all TP and FP encompasses all of the actual results, while the sum of all TP and FN encompasses all of the predicted results, one can deduce that precision means the percentage of the results which are relevant, and on the other hand, recall refers to the percentage of total relevant results correctly classified by the classifier (Pedregosa et al., 2011).

However, COCO (Lin et al., 2014) metrics which is used in this experiment uses mean average precision (mAP) with Intersection over Union (IoU) between 0.5 and 0.95 with steps of 0.05 (AP@IoU = 0.5:0.05:0.95). An average precision (AP) is calculated from the area under a precision-recall curve. Whereas mAP takes the average of AP at each IoU from 0.5 to 0.95 with steps of 0.05 (total 10 APs).

COCO also uses mean average recall (mAR) with the same IoU values, but instead of using the area under a precision-recall curve, it uses a recall-IoU curve.

5.2.2 Accuracy and Loss

Accuracy is also used as a statistical measure of the correct identification or exclusion of a condition by a binary classification test. That is, the accuracy is the proportion of true results (both TP and TN) among the total number of cases examined (Metz, 1978).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

In the DeepFashion paper, the top N accuracy metrics was used for comparison with other detection models. Top N accuracy is essentially the same as accuracy but with all the top N true results are considered among the total number of cases examined.

For the first stage of Faster R-CNN (Szegedy et al., 2015), the RPN loss function is used:

$$L\left(\left\{\mathrm{p}_i\right\}, \left\{\mathrm{t}_i\right\}\right) = \frac{1}{\mathrm{N}_{cls}} \sum_i \mathrm{L}_{cls}\left(\mathrm{p}_i, \mathrm{p}_{*i}\right) + \lambda \frac{1}{\mathrm{N}_{loc}} \sum_i \mathrm{p}_{*i} \mathrm{L}_{loc}\left(\mathrm{t}_{i, \mathrm{t}_{*i}}\right)$$

From the above equation, i is the index of the anchor in the mini-batch. The classification loss (can also be called objectness loss as it is classifying a binary choice of object vs not object) $\mathrm{L}_{cls}\left(\mathrm{p}_i, \mathrm{p}_{*i}\right)$ is the log loss

over two classes (object vs not object). p_i is the output score from the classification branch for anchor i, and p_i* is the groundtruth label (1 or 0).

The localization loss $L_{loc}\left(t_i, t_{*i}\right)$ is activated only if the anchor actually contains an object i.e., the groundtruth p_i* is 1. The term t_i is the output prediction of the regression layer and consists of 4 variables [t_x, t_γ, tw, th]. The regression target t_i* is calculated as:

$$t_x^* = \frac{\left(x^* - x_a\right)}{w_a}, \; t_y^* = \frac{\left(y^* - y_a\right)}{h_a}, \; t_w^* = \log\left(w^* / \; w_a\right), \; t_h^* = \log\left(h^* / \; h_a\right)$$

Here, x, y, w, and h correspond to the (x, y) coordinates of the top-left coordinate and the height (h) and width (w) of the box. x*, x_a stands for the coordinates of the anchor box and its corresponding groundtruth bounding box.

At the second stage, another loss function for the box classifier is used, it was inherited from its predecessor - Fast R-CNN (Girshick, 2015):

The classification loss $L_{cls}(p,u) = -\log\left(p_u\right)$ is combined with the localization loss $L_{loc}(t^u,v) = \sum_{i \in \{x,y,w,h\}} \text{smooth}_{L1}\left(t_i^u - v_i\right)$ where (x,y) stands for the top-left corner, w = weight and h = height of the bounding box, to form a joint multi-task loss.

However, there is a $\mathbb{1}[u>=1] = \begin{cases} 1 & if \; u \geq 1 \\ 0 & otherwise \end{cases}$ where the localization loss is ignored when the background classes have no groundtruth boxes, as shown here, the overall function:

$$L(p,u,t^u,v) = L_{cls}(p,u) + \lambda[u \geq 1] L_{loc}(t^u,v)$$

5.3 Results and Evaluation

Through Tensorboard and results from manual coding for custom evaluation metrics that were not included in the COCO evaluation protocol, the results are revealed in the following subsections. The final training and evaluation process took about 4 days and 12 hours with the aforementioned software and hardware setup.

5.3.1 Detection

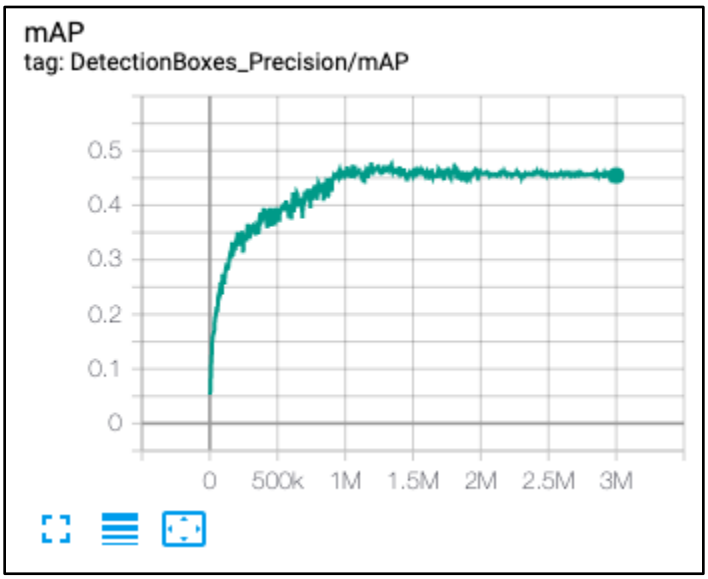


Figure 5.3: Mean average precision (mAP) of the detection boxes of this experiment.

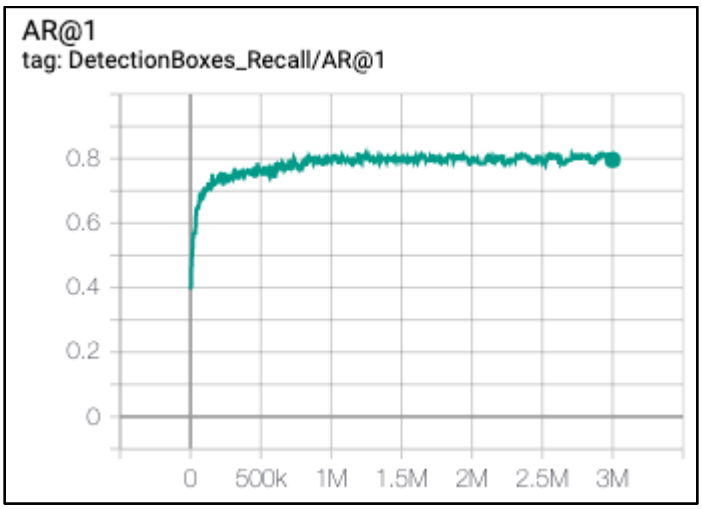


Figure 5.4: Mean average recall (mAR) of the detection boxes of this experiment.

From Figure 5.3 and 5.4, the final mAP of the experiment from the beginning to 3,000,000 steps or 12 epoch is at 0.4546, whereas mAR is at 0.7961.

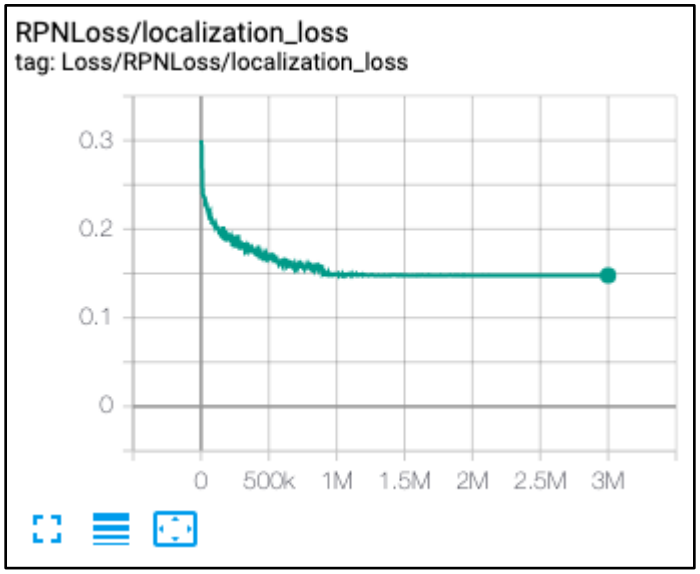


Figure 5.5: Localization loss of the Region Proposal Network.

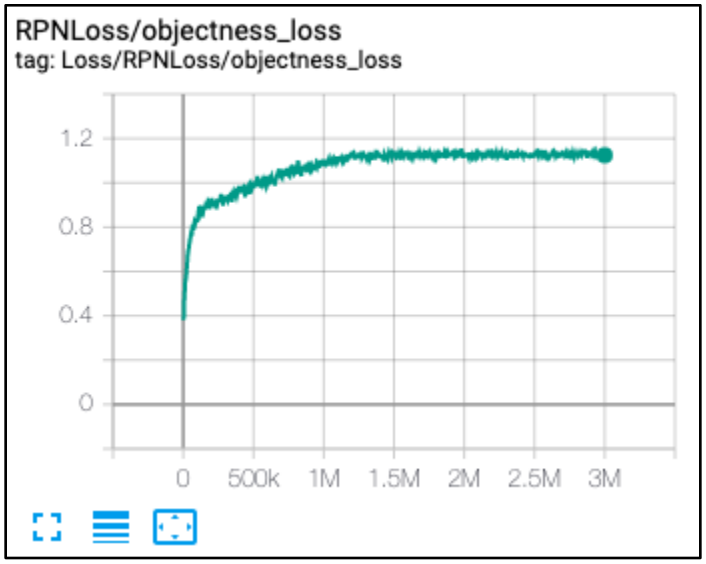


Figure 5.6: Objectness loss of the Region Proposal Network.

Results from Figure 5.5 deduce the final value of localization loss from the beginning to 3,000,000 steps or 12 epoch at 0.1477. Figure 5.6 shows the final value of the objectness loss at 1.124.

Figure 5.7: Detection left vs groundtruth right at 817,145 steps.

Figure 5.8: Detection left vs groundtruth right at 3,000,000 steps.

Figure 5.7 and 5.8 show comparisons between detections at 817,145 steps and 3,000,000 steps; the earlier detections boxes show more differences from their respective groundtruths.

5.3.2 Classification

The top-N accuracy after 3,000,000 steps are also measured:

Top-N	Accuracy (%)
1	73.86
2	81.69
3	83.71
4	83.71
5	83.71

Table 5.1: Top-N accuracy of the experiment where 1<=N<=5.

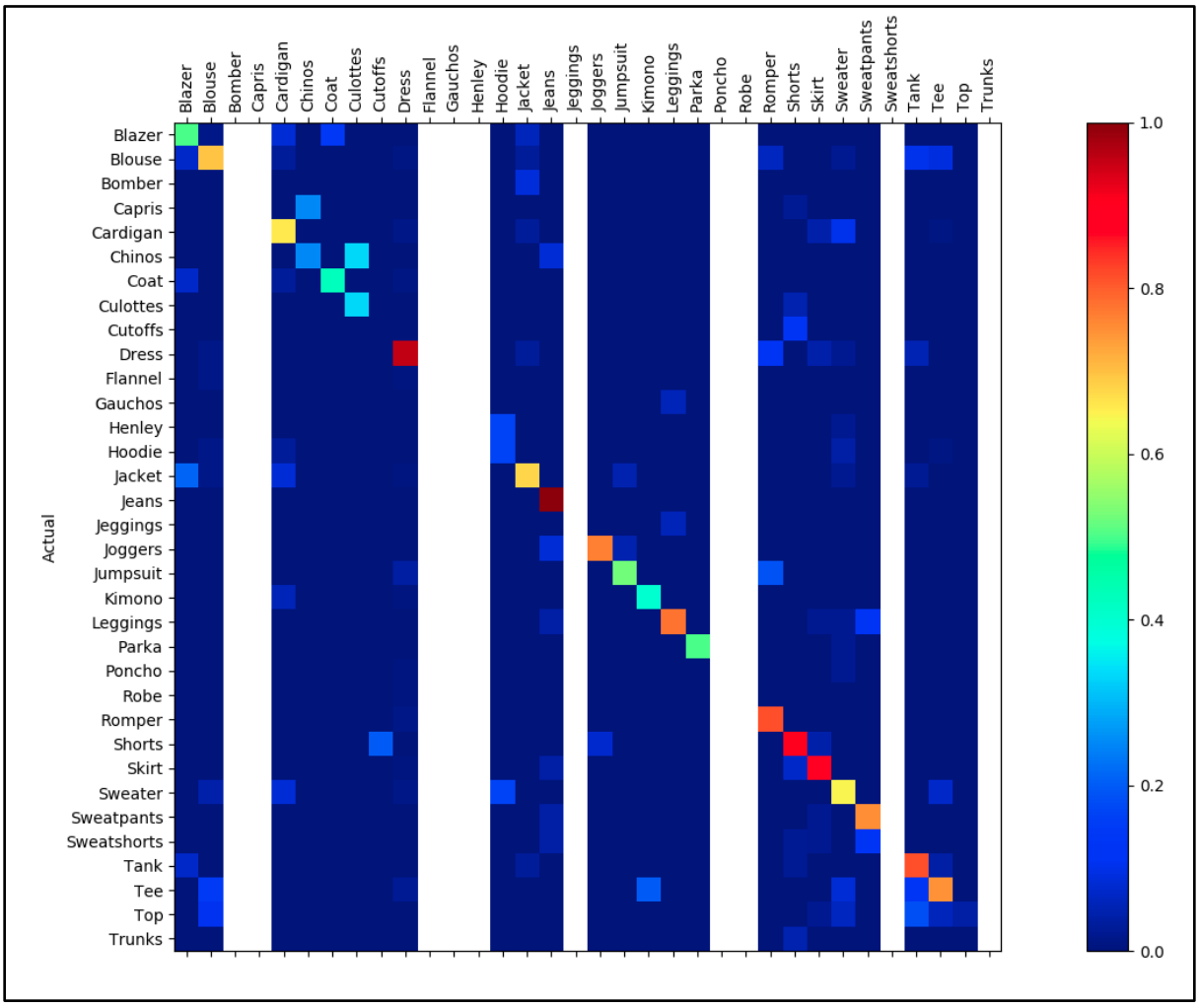


Figure 5.9: Confusion matrix of the experiment.

At Figure 5.9, a confusion matrix is drawn after 3,000,000 steps or 12 epoch. The colormap used was a heatmap, where the warmer the color is, the higher the value, and vice-versa. A version of the matrix with numbers is attached in Appendix A.

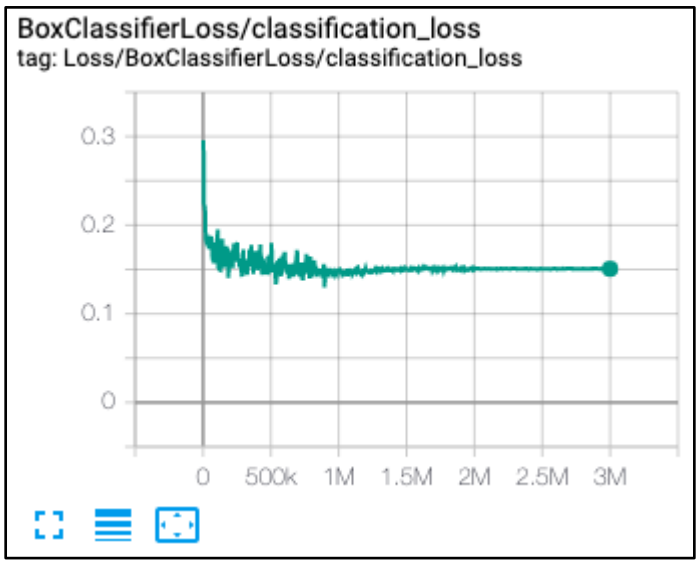


Figure 5.10: Classification loss of the box classifier.

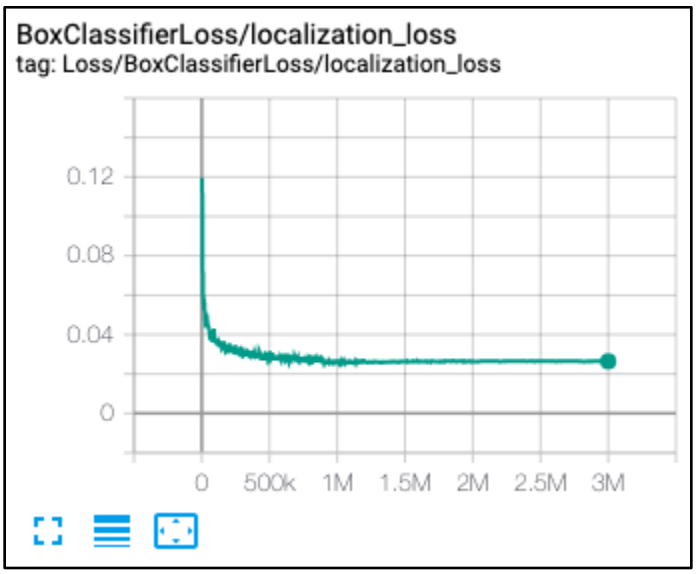


Figure 5.11: Localization loss of the box classifier.

Figure 5.10 and 5.11 show classification and localization loss of the box classifier, respectively, the former is at 0.1507, whereas the latter at 0.0264.

Figure 5.12: Classification results vs groundtruth at 817,145 steps.

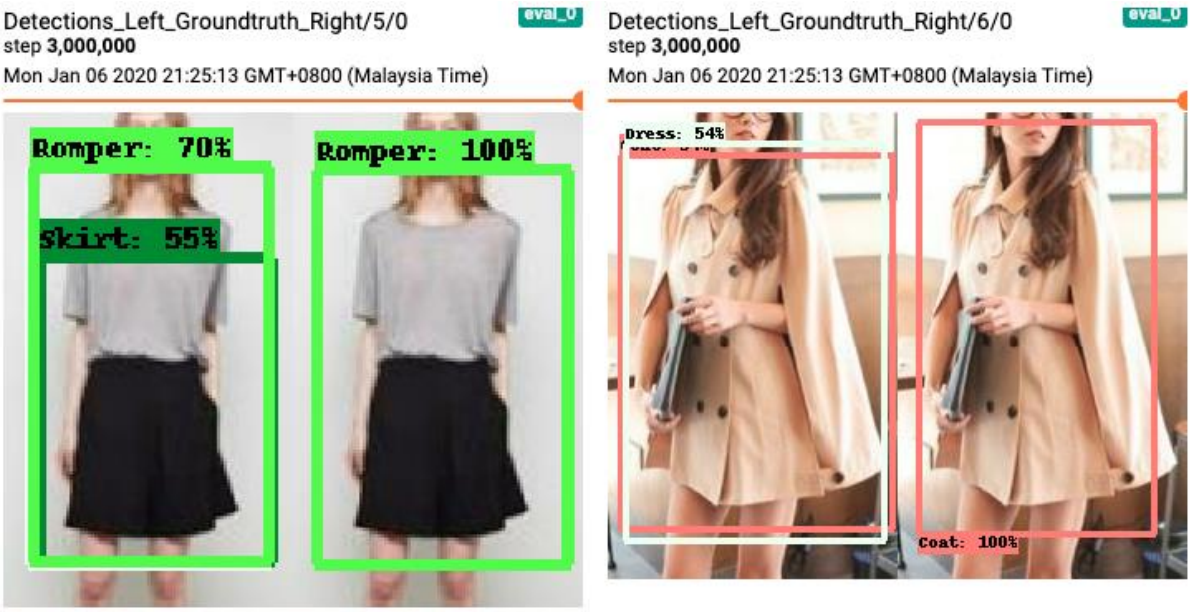


Figure 5.13: Classification results vs groundtruth at 3,000,000 steps.

Figure 5.13 and 5.14 show comparisons between classification results at 817,415 steps and 3,000,000 steps; the former being less accurate than the latter.

5.4 Real-world Results

From a brief testing with 30 randomly picked images from the Internet with similar resolutions as the training and evaluation sets, only 6 gives correct classification results (one detection box each). That gives a 20% top-1 accuracy, but due to the small sample size, it was merely a rough estimate.

Testing with real-world amatuer images (non-studio lit or noisy), the model returns unsable results.

5.5 Summary

In Section 5.1, the setup and configuration for the experiment is introduced and detailed. Chronologically, it involves setting up a runtime environment, processing the dataset, and configuring the API configuration files.

In Section 5.2, the evaluation metrics used were introduced and explained, which includes a confusion matrix, the use of precision and recall (as well as mean average precision and mean average recall), accuracy (also top-n accuracy), and loss (which has 4 loss functions covered - the RPN classification loss, RPN localization loss, classifier classification loss, and the classifier localization loss).

In Section 5.3 and 5.4, the evaluation results were revealed. The results were also separated into detection and classification. These results shall be discussed in Chapter 6.

Chapter 6

Discussion

6 Discussion

This chapter involves discussion about the achievements, issues, solutions, limitations and future improvements of this research.

6.1 Achievements

Through this research, a series of tasks set were successful, they include setting up to use the Tensorflow's Object Detection API (Abadi et al., 2016), converting the DeepFashion (Liu et al., 2016) dataset to a standard format - the TFRecord format, manual coding for further evaluation tasks (top-n accuracy and confusion matrix), and most importantly, a clothing image retrieval model was trained and achieved admirable results. The results shall be discussed as follows:

Table 6.1 shows the comparisons between the different models that use DeepFashion dataset for clothing image retrieval on category classification. From the table, Faster R-CNN (Szegedy et al., 2015) with Inception V2 (Szegedy et al. 2016) ranks first in its top-3 accuracy while ranks third in its top-5 accuracy. This shows that for practical usage, which is when top-n where n is at the smallest possible (from this table, smallest possible is 3), Faster R-CNN with Inception V2 weighs higher in real world practical usage. Faster R-CNN has also outperformed WTBI (Chen et al., 2012) by 40 percent and DARN (Huang et al., 2015) by 24 percent at top-3 accuracy. In comparison with FashionNet from the DeepFashion team, Faster R-CNN with Inception V2 outperforms by 1.13 percent at top-3 accuracy.

Neural Network Model	**Clothing Image Retrieval (Category Classification)**	
	Top-3 Accuracy (%)	**Top-5 Accuracy (%)**
WTBI	43.73	66.26
DARN	59.48	79.58
FashionNet+100	47.38	70.57
FashionNet+500	57.44	77.39
FashionNet+Joints	72.80	81.52
FashionNet+Poselets	75.34	84.87
FashionNet	82.58	**90.17**
SSD Resnet50	83.46	83.46
Faster R-CNN Inception V2	**83.71**	83.71

Table 6.1: Comparisons of different models for clothing image retrieval on category classification.

Table 6.2 shows the comparisons between Faster R-CNN with Inception V2 and SSD with Resnet50 (Liu et al., 2016) (He et al. 2015) for clothing image retrieval on category classification, with Top-1 accuracy included. Here, even though Faster R-CNN with Inception V2 triumphs at top-3 and top-5 accuracy, SSD with Resnet50 outperforms the former by a mere 0.51 percent.

Neural Network Model	**Clothing Image Retrieval (Category Classification)**		
	Top-1 Accuracy (%)	**Top-3 Accuracy (%)**	**Top-5 Accuracy (%)**
SSD Resnet50	**74.37**	83.46	83.46
Faster R-CNN Inception V2	73.86	**83.71**	**83.71**

Table 6.2: Comparisons between Faster R-CNN Inception V2 and SSD Resnet50 for clothing image retrieval on category classification, with Top-1 accuracy included.

Furthermore, through the confusion matrix that was revealed in Section 5.3.2, it was obvious that the model was consistent in its results, that the 'heat' was mainly located in the diagonals, which means that the model does not appear to be too 'confused' - most results are accurate and true positives.

Regarding detections, as shown in Section 5.3.1, the constant increase of mAP and mAR values till their convergence and the further resemblance of the bounding boxes of the predictions against the groundtruths proved that the model was indeed learning to detect clothing better.

6.2 Issues and Solutions

The API is a convenient method to have less involvement in coding in object detection research, however, the API comes with few numbers of tradeoffs:

First, the need for the conversion of dataset to a standardized format of TFRecord, as described in Section 3.1.1, may increase the complexity of the experiment.

Two, the lack of flexibility when it comes to customization of evaluation matrices, as the only ones that are available such as PASCAL VOC (Everingham et al., 2010), COCO (Lin et al., 2014), and Open Images (Kuznetsova et al., 2018) are limited and not customizable, for instance, this paper's experiment uses the COCO detection metric which does not include classification top-N accuracy and confusion matrix, the former which must be acquired for comparison with the DeepFashion (Liu et al., 2016) paper, which uses Top-3 accuracy.

To resolve these issues, involvement in coding ironically, was needed. Though the API was able to save time through less coding, it lacks a degree of flexibility, hence the need of custom codes. Ergo, the parts involving conversion of dataset to TFRecord and evaluation metrics such as top-n accuracy and confusion matrix were all manually coded.

In terms of the training process, it is common that one is not able to know the values of learning rate needed. To solve this problem, hyperparameter optimization can be used. There are multiple methods of hyperparameter optimization, but a method called cyclical learning rates (CLR) (Smith, 2017) was ultimately chosen for its effectiveness without the use of too much resources unlike grid search (Pontes et al., 2016). Through CLR, a set of effective learning rates for different schedule/time scale through different number of steps was found.

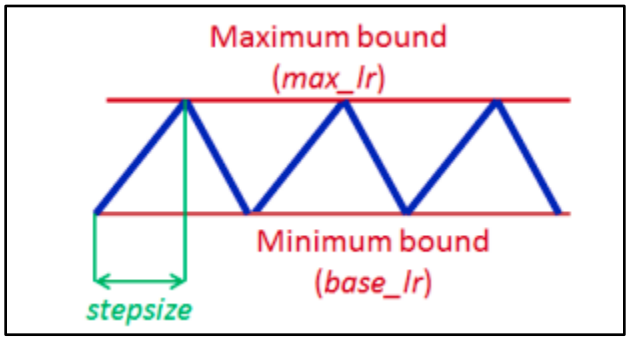


Figure 6.1: The triangular learning rate policy of CLR. The blue lines represent learning rate values going up and down between maximum and minimum bounds.

6.3 Limitations and Future Improvements

As mentioned in Section 5.1.3, for the evaluation phase, one of every 50 images were chosen for evaluation. That sums up to 2 percent or 1,000 images of the total evaluation images of 50,000 provided by DeepFashion. This decision was ultimately made due to hardware and resource limitations, as these 1,000 images have already taken about 15 minutes for each evaluation phase (which happens about every 10 minutes through the training phase). To overcome this limitation, a future continuation of this research could use better hardware and more resources.

In Section 5.4, it was explained that the model performed rather poorly on real-time images. After further inspection, it was discovered that the DeepFashion dataset's evaluation images and training images were quite similar, and lack variation. Furthermore, the confusion matrix in Section 5.3.2 also reveals that only few categories were consistent in their results. This may be due to its imbalance in attribute distribution (Zakizadeh et al., 2018). For improvements, the successor of this research shall use a better dataset such as DeepFashion2 (Ge et al., 2019).

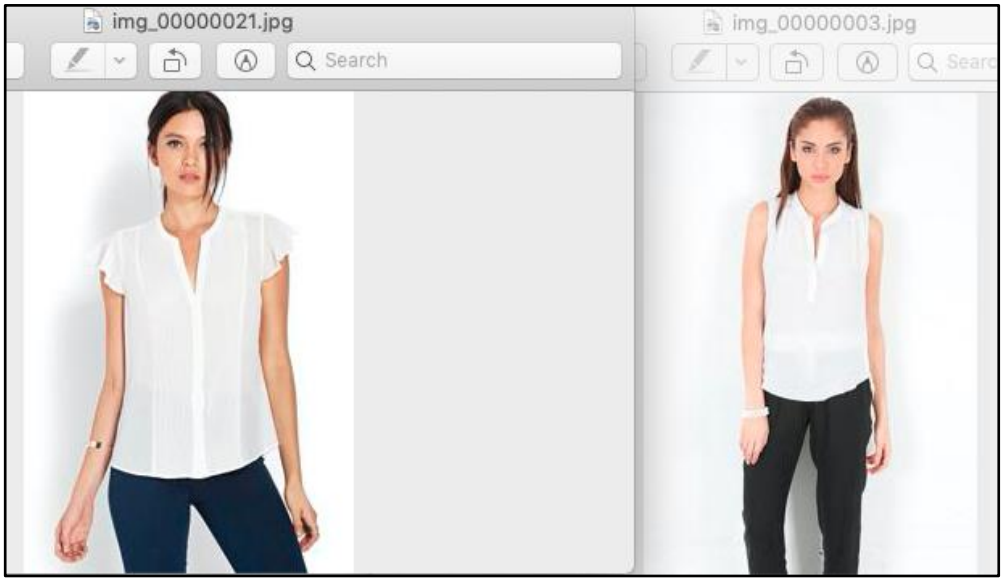


Figure 6.2: The image on the left is from the training set while image on the right is from the evaluation set. Many clothing from the evaluation and training set of DeepFashion appears very similar.

Furthermore, unlike FashionNet which was designed specifically for clothing image retrieval, Faster R-CNN with Inception V2 does not consider body landmarks and poselets as well as FashionNet, though Faster R-CNN with Inception V2 scored better in accuracy than FashionNet, it could be due to FashionNet being a rather dated model. Thus, a modern object detection model with features to detection specific body landmarks and poselets could make an even better clothing image retrieval model.

6.4 Summary

This chapter reveals few notable information, such as:

Firstly, Faster R-CNN with Inception V2 topped all other models in top-3 accuracy for clothing categorical classification. However, it scored slightly less for top-1 and top-5 accuracy when compared with SSD with ResNet50 and FashionNet, respectively.

Secondly, Tensorflow's Object Detection API lacks flexibility in custom evaluation metrics, further manual coding is required to add more metrics.

Thirdly, DeepFashion dataset lacks variation and differences between the training set and evaluation set, it also has an unbalanced attribute distribution. The successor of this research should use DeepFashion2.

Finally, Faster R-CNN with Inception V2 could perform even better if it considers body landmarks and poselets like FashionNet when performing clothing image retrievals.

Chapter 7

Conclusion

7 Conclusion

The domain of fashion or clothing image retrieval remains a challenging task. As a newer model than FashionNet (Liu et al., 2016), Faster R-CNN (Szegedy et al., 2015) with Inception V2 (Szegedy et al. 2016) performed better than the former albeit without features specific to body landmarks and poselets prediction. The Tensorflow Object Detection API (Abadi et al., 2016) was also ultimately chosen for the task as it is a mature platform with an advantage of having less coding required, thus time and cost saving. However, it is inflexible when it comes to using custom evaluation metrics, thus minimal coding was done to compensate for this inflexibility.

Transfer learning was also chosen as the preferred method as it requires much less resources than training from scratch. Method of transfer learning used was to use frozen convolution layers and train with new convolution layers on top of them, as the dataset is large and the features can be quite similar to the original COCO (Lin et al., 2014) dataset.

Due to the use of CLR, learning rates were set accordingly without the need of guessing. The regularizer was also changed from the default L2 to L1 due to it being better suited for larger dataset like DeepFashion.

With a modern model like Faster R-CNN with Inception V2 as its base model, the performance was quite admirable as even without special predictors for body landmarks and poselets, it was able to beat FashionNet by 1.13 percent at top-3 accuracy. With the availability of many other pre-trained models in Tensorflow's Object Detection API, many other modern detection models could also be tested for clothing image retrieval, which would be able to bring more improvement to this domain in the future with lesser effort.

References

Abadi, M. et al., 2016. TensorFlow: A System for Large-Scale Machine Learning. 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16). November 2016 USENIX Association, Savannah, GA, pp. 265–283.

Bengio, Y., 2012. Practical Recommendations for Gradient-Based Training of Deep Architectures. In: Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 437–478.

Bottou, L., 2010. Large-Scale Machine Learning with Stochastic Gradient Descent. In: Lechevallier, Y. and Saporta, G., (eds.) Proceedings of COMPSTAT'2010. Physica-Verlag HD, Heidelberg, pp. 177–186.

Bradski, G., 2000. The OpenCV Library. Dr. Dobb's Journal of Software Tools.

Brownlee, J., 2019, A Gentle Introduction to Convolutional Layers for Deep Learning Neural Networks [Online]. Available at: https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/ [Accessed: 21 August 2019].

Chakrabarty, A., 2010. An Investigation of Clustering Algorithms and Soft Computing Approaches for Pattern Recognition. Doctoral dissertation, Assam University.

Chen, H., Gallagher, A. and Girod, B., 2012. Describing clothing by semantic attributes. In: Fitzgibbon, A. et al., (eds.) Computer vision – ECCV 2012. Lecture notes in computer science. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 609–623.

Chollet, F., 2018. Deep Learning With Python 1st ed., Manning Publications, Shelter Island, New York.

cs231n, Convolutional Neural Networks (CNNs / ConvNets) [Online]. Available at: http://cs231n.github.io/convolutional-networks/ [Accessed: 23 August 2019b].

Davis, J. and Goadrich, M., 2006. The relationship between Precision-Recall and ROC curves. ICML '06: Proceedings of the 23rd international conference on Machine learning. 2006 ACM, Pittsburgh, Pennsylvania, pp. 233--240.

Everingham, M. et al., 2010. The Pascal Visual Object Classes (VOC) Challenge. International Journal of Computer Vision, 88(2), pp.303–338. Available at: http://dblp.uni-trier.de/db/journals/ijcv/ijcv88.html#EveringhamGWWZ10.

Ge, Y. et al., 2019. DeepFashion2: A Versatile Benchmark for Detection, Pose Estimation, Segmentation and Re-Identification of Clothing Images. CoRR, abs/1901.07973.

Girshick, R., Donahue, J., Darrell, T. and Malik, J., 2014. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. 2014 IEEE Conference on Computer Vision and Pattern Recognition. June 2014 IEEE.

Girshick, R., 2015. Fast R-CNN. 2015 IEEE International Conference on Computer Vision (ICCV). 7 December 2015 IEEE, pp. 1440–1448.

He, K., Gkioxari, G., Dollar, P. and Girshick, R., 2017. Mask R-CNN. 2017 IEEE International Conference on Computer Vision (ICCV). 22 October 2017 IEEE, pp. 2980–2988.

He, K., Zhang, X., Ren, S. and Sun, J., 2015. Deep Residual Learning for Image Recognition. CoRR, abs/1512.03385.

Hopfield, J.J., 1982. Neural networks and physical systems with emergent collective computational abilities. Proceedings of the National Academy of Sciences of the United States of America, 79(8), pp.2554–2558.

Howard, W.R., 2007. Pattern Recognition and Machine Learning20072Christopher M. Bishop.Pattern Recognition and Machine Learning . Heidelberg, Germany: Springer 2006. i-xx, 740 pp., ISBN: 0-387-31073-8 $74.95 Hardcover. Kybernetes, 36(2), pp.275–275.

Huang, J., Feris, R., Chen, Q. and Yan, S., 2015. Cross-Domain Image Retrieval with a Dual Attribute-Aware Ranking Network. 2015 IEEE International Conference on Computer Vision (ICCV). 7 December 2015 IEEE, pp. 1062–1070.

Huang, J. et al., 2017. Speed/Accuracy Trade-Offs for Modern Convolutional Object Detectors. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). July 2017 IEEE.

Innovation Enterprise, Big Data Hits the Runway: How Big Data is Changing the Fashion Industry [Online]. Available at: https://channels.theinnovationenterprise.com/articles/8230-big-data-hits-the-runway-how-big-data-is-changing-the-fashion-industry [Accessed: 22 August 2019a].

Jain, A.K., Mao, J. and Mohiuddin, K., 1996. Artificial Neural Networks: A Tutorial. IEEE Computer, 29, pp.31–44.

Jia, Y. et al., 2014. Caffe: Convolutional Architecture for Fast Feature Embedding. arXiv preprint arXiv:1408.5093.

John J. et al., 2019. Review Paper on Object Detection using Deep Learning- Understanding different Algorithms and Models to Design Effective Object Detection Network. International Journal for Research in Applied Science & Engineering Technology (IJRASET), Volume 7 Issue III.

Kepka, J., 1994. The current approaches in pattern recognition. Kybernetika, 30, pp.159–176.

Kiapour, M.H. et al., 2015. Where to buy it: matching street clothing photos in online shops. 2015 IEEE International Conference on Computer Vision (ICCV). 7 December 2015 IEEE, pp. 3343–3351.

Kohavi, R., 2001. A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. International Joint Conference on Arti cial Intelligence (IJCAI), 1995, 14.

Krizhevsky, A., Sutskever, I. and Hinton, G.E., 2012. ImageNet classification with deep convolutional neural networks. Communications of the ACM, 60(6), pp.84–90.

Kuznetsova, A. et al., 2018. The Open Images Dataset V4: Unified image classification, object detection, and visual relationship detection at scale. CoRR, abs/1811.00982. Available at: http://dblp.uni-trier.de/db/journals/corr/corr1811.html#abs-1811-00982.

LeCun, Y. et al., 1989. Backpropagation applied to handwritten zip code recognition. Neural Computation, 1(4), pp.541–551.

Lecun, Y., Bottou, L., Bengio, Y. and Haffner, P., 1998. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), pp.2278–2324.

Lin, M., Chen, Q. and Yan, S., 2013. Network In Network. CoRR, abs/1312.4400.

Lin, T.-Y. et al., 2014. Microsoft COCO: Common Objects in Context. In: Computer Vision – ECCV 2014. Springer International Publishing, pp. 740–755.

Liu, Z. et al., 2016. DeepFashion: Powering Robust Clothes Recognition and Retrieval with Rich Annotations. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 27 June 2016 IEEE, pp. 1096–1104.

Luo, P., Wang, X. and Tang, X., 2013. Pedestrian parsing via deep decompositional network. 2013 IEEE International Conference on Computer Vision. 1 December 2013 IEEE, pp. 2648–2655.

Marcelino P., 2018. Transfer learning from pre-trained models. [Online] Towards Data Science.

Available at: https://towardsdatascience.com/transfer-learning-from-pre-trained-models-f2393f124751 [Accessed: 21 August 2019].

McCulloch, W.S. and Pitts, W., 1990. A logical calculus of the ideas immanent in nervous activity. 1943. Bulletin of Mathematical Biology, 52(1–2), p.99–115; discussion 73.

Metz, C.E., 1978. Basic principles of ROC analysis. Seminars in Nuclear Medicine, 8(4), pp.283–298. Available at: http://dx.doi.org/10.1016/s0001-2998(78)80014-2.

Nandhakumar, N. and Aggarwal, J.K., 1985. The artificial intelligence approach to pattern recognition—a perspective and an overview. Pattern recognition, 18(6), pp.383–389.

Nayak, S., 2018, Understanding AlexNet [Online]. Available at: https://www.learnopencv.com/understanding-alexnet/ [Accessed: 21 August 2019].

Nicholson, C., A Beginner's Guide to Neural Networks and Deep Learning [Online]. Available at: https://skymind.ai/wiki/neural-network.html [Accessed: 22 August 2019].

Nielsen, M., 2019, How the backpropagation algorithm works [Online]. Available at: http://neuralnetworksanddeeplearning.com/chap2.html [Accessed: 21 August 2019].

Nievergelt, J., 1969. R69-13 Perceptrons: An Introduction to Computational Geometry. IEEE Transactions on Computers, C-18(6), pp.572–572.

Pedregosa, F. et al., 2011. Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12(Oct), pp.2825--2830.

Pontes, F.J. et al., 2016. Design of experiments and focused grid search for neural network parameter optimization. Neurocomputing, 186, pp.22–34. Available at: http://dblp.uni-trier.de/db/journals/ijon/ijon186.html#PontesABPF16.

Programmable Web, Deepomatic Fashion Apparel Detection API [Online]. Available at: https://www.programmableweb.com/api/deepomatic-fashion-apparel-detection [Accessed: 21 August 2019c].

Quaknine A. 2018. Review of Deep Learning Algorithms for Object Detection. [Online] Medium. Available at: https://medium.com/zylapp/review-of-deep-learning-algorithms-for-object-detection-c1f3d437b852 [Accessed: 21 August 2019].

Rawat, W. and Wang, Z., 2017. Deep convolutional neural networks for image classification: A comprehensive review. Neural Computation, 29(9), pp.2352–2449.

Redmon, J., Divvala, S., Girshick, R. and Farhadi, A., 2016. You Only Look Once: Unified, Real-Time Object Detection. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). June 2016 IEEE.

Ren, S., He, K., Girshick, R. and Sun, J., 2017. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. IEEE transactions on pattern analysis and machine intelligence, 39(6), pp.1137–1149.

Russakovsky, O. et al., 2015. ImageNet large scale visual recognition challenge. International journal of computer vision, 115(3), pp.211–252.

Schalkoff, R.J., 2007. Pattern Recognition. In: Wah, B.W., (ed.) Wiley encyclopedia of computer science and engineering. John Wiley & Sons, Inc., Hoboken, NJ, USA.

Schmidhuber, J., 2015. Deep learning in neural networks: an overview. Neural Networks, 61, pp.85–117.

Shorten, C., 2019, Introduction to ResNets [Online]. Available at:

https://towardsdatascience.com/introduction-to-resnets-c0a830a288a4 [Accessed: 21 August 2019].

Simonyan, K. and Zisserman, A., 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. 2015 arXiv.

Skalski, P., 2018, Preventing Deep Neural Network from Overfitting [Online]. Available at: https://towardsdatascience.com/preventing-deep-neural-network-from-overfitting-953458db800a [Accessed: 21 August 2019].

Smith, L.N., 2017. Cyclical Learning Rates for Training Neural Networks. WACV. 2017 IEEE Computer Society, pp. 464–472.

Szegedy, C. et al., 2014. Going Deeper with Convolutions. CoRR, abs/1409.4842.

Szegedy, C. et al., 2015. Rethinking the Faster R-CNN for Computer Vision. CoRR, abs/1512.00567.

Szegedy, C. et al., 2016. Rethinking the Inception Architecture for Computer Vision. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). June 2016 IEEE.

TensorFlow (2019). ObjectDetection API not suitable for tf 2.0.0-alpha0. [online] GitHub. Available at: https://github.com/tensorflow/models/issues/6423 [Accessed: 21 August 2019].

TensorFlow (2019). Tensorflow detection model zoo. [online] GitHub. Available at: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md [Accessed: 21 August 2019].

Tsang, S.-H., 2018a, Review: GoogLeNet (Inception v1)— Winner of ILSVRC 2014 (Image Classification) [Online]. Available at: https://medium.com/coinmonks/paper-review-of-googlenet-inception-v1-winner-of-ilsvlc-2014-image-classification-c2b3565a64e7 [Accessed: 22 August 2019].

Tsang, S.-H., 2018b, Review: VGGNet — 1st Runner-Up (Image Classification), Winner (Localization) in ILSVRC 2014 [Online]. Available at: https://medium.com/coinmonks/paper-review-of-vggnet-1st-runner-up-of-ilsvlc-2014-image-classification-d02355543a11 [Accessed: 22 August 2019].

Werbos, P., 1974. Beyond regression : new tools for prediction and analysis in the behavioral sciences /. Undergraduate thesis,

Wired, 2019, Here's how Nike, Alibaba and Walmart are reinventing retail [Online]. Available at: https://www.wired.co.uk/article/future-of-retail [Accessed: 22 August 2019].

Yang, M. and Yu, K., 2011. Real-time clothing recognition in surveillance videos. 2011 18th IEEE International Conference on Image Processing. 11 September 2011 IEEE, pp. 2937–2940.

Yiu, T., 2019, Understanding Neural Networks [Online]. Available at: https://towardsdatascience.com/understanding-neural-networks-19020b758230 [Accessed: 22 August 2019].

Zadeh, L.A., 1965. Fuzzy sets. Information and Control, 8(3), pp.338–353.

Zakizadeh, R., Sasdelli, M., Qian, Y. and Vazquez, E., 2018. Improving the Annotation of DeepFashion Images for Fine-grained Attribute Recognition. CoRR, abs/1807.11674. Available at: http://dblp.uni-trier.de/db/journals/corr/corr1807.html#abs-1807-11674.

Zhou, X., 2018. Understanding the Convolutional Neural Networks with Gradient Descent and Backpropagation. Journal of Physics: Conference Series, 1004, p.012028.

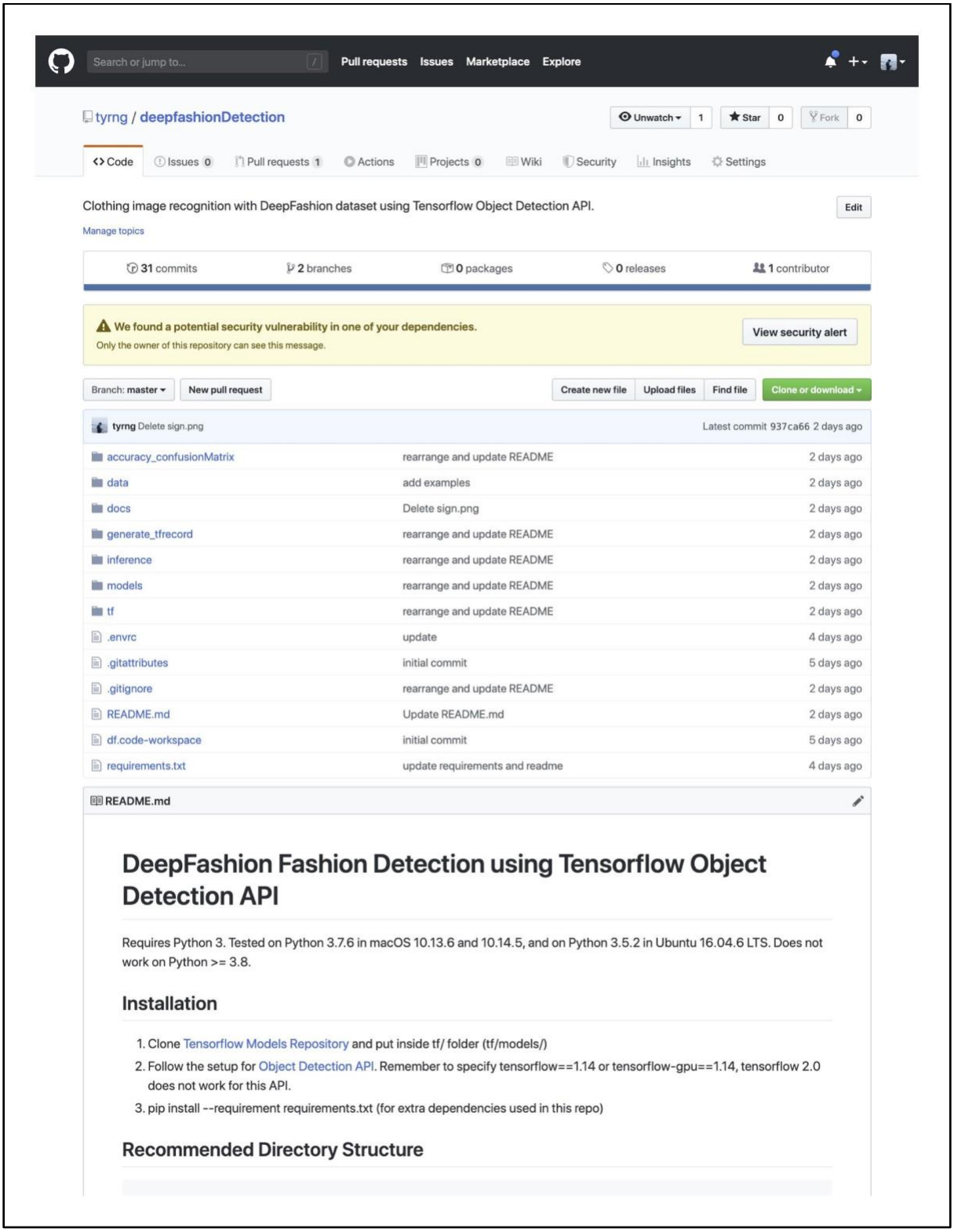
Appendices

Appendix A: Confusion Matrix at 3,000,000 steps (With Numbers)

Actual	Blazer	Blouse	Bomber	Capris	Cardigan	Chinos	Coat	Culottes	Cutoffs	Dress	Flannel	Gauchos	Henley	Hoodie	Jacket	Jeans
Blazer	0.384615	0.009709	NaN	NaN	0.075758	0.000000	0.066667	0.0	0.000	0.000000	NaN	NaN	NaN	0.000000	0.145455	0.000000
Blouse	0.076923	0.524272	NaN	NaN	0.045455	0.000000	0.000000	0.0	0.000	0.039062	NaN	NaN	NaN	0.000000	0.036364	0.000000
Bomber	0.000000	0.000000	NaN	NaN	0.000000	0.000000	0.000000	0.0	0.000	0.000000	NaN	NaN	NaN	0.000000	0.054545	0.000000
Capris	0.000000	0.000000	NaN	NaN	0.000000	0.030909	0.000000	0.0	0.000	0.000000	NaN	NaN	NaN	0.000000	0.000000	0.000000
Cardigan	0.076923	0.038835	NaN	NaN	0.500000	0.000000	0.000000	0.0	0.000	0.023438	NaN	NaN	NaN	0.090909	0.054545	0.000000
Chinos	0.000000	0.000000	NaN	NaN	0.000000	0.181818	0.000000	0.2	0.000	0.000000	NaN	NaN	NaN	0.000000	0.000000	0.111111
Coat	0.076923	0.009709	NaN	NaN	0.045455	0.000000	0.333333	0.0	0.000	0.007812	NaN	NaN	NaN	0.000000	0.036364	0.000000
Culottes	0.000000	0.000000	NaN	NaN	0.000000	0.000000	0.000000	0.2	0.000	0.000000	NaN	NaN	NaN	0.000000	0.000000	0.037037
Cutoffs	0.000000	0.000000	NaN	NaN	0.000000	0.000000	0.000000	0.0	0.375	0.000000	NaN	NaN	NaN	0.000000	0.000000	0.000000
Dress	0.038462	0.097087	NaN	NaN	0.015152	0.000000	0.066667	0.0	0.000	0.753906	NaN	NaN	NaN	0.000000	0.018182	0.000000
Flannel	0.000000	0.019417	NaN	NaN	0.000000	0.000000	0.000000	0.0	0.000	0.007812	NaN	NaN	NaN	0.000000	0.000000	0.000000
Gauchos	0.000000	0.000000	NaN	NaN	0.000000	0.000000	0.000000	0.0	0.000	0.000000	NaN	NaN	NaN	0.000000	0.000000	0.000000
Henley	0.000000	0.000000	NaN	NaN	0.000000	0.000000	0.000000	0.0	0.000	0.000000	NaN	NaN	NaN	0.090909	0.000000	0.000000
Hoodie	0.000000	0.009709	NaN	NaN	0.015152	0.000000	0.000000	0.0	0.000	0.003906	NaN	NaN	NaN	0.181818	0.000000	0.000000
Jacket	0.346154	0.019417	NaN	NaN	0.151515	0.000000	0.133333	0.0	0.000	0.007812	NaN	NaN	NaN	0.090909	0.454545	0.000000
Jeans	0.000000	0.000000	NaN	NaN	0.000000	0.000000	0.000000	0.0	0.000	0.000000	NaN	NaN	NaN	0.000000	0.000000	0.888889
Jeggings	0.000000	0.000000	NaN	NaN	0.000000	0.000000	0.000000	0.0	0.000	0.000000	NaN	NaN	NaN	0.000000	0.000000	0.000000
Joggers	0.000000	0.000000	NaN	NaN	0.000000	0.090909	0.000000	0.0	0.000	0.000000	NaN	NaN	NaN	0.000000	0.000000	0.111111
Jumpsuit	0.000000	0.000000	NaN	NaN	0.015152	0.000000	0.066667	0.0	0.000	0.054688	NaN	NaN	NaN	0.000000	0.000000	0.000000
Kimono	0.000000	0.000000	NaN	NaN	0.030303	0.000000	0.000000	0.0	0.000	0.003906	NaN	NaN	NaN	0.000000	0.000000	0.000000
Leggings	0.000000	0.000000	NaN	NaN	0.000000	0.000000	0.000000	0.0	0.000	0.000000	NaN	NaN	NaN	0.000000	0.000000	0.074074
Parka	0.000000	0.000000	NaN	NaN	0.015152	0.000000	0.000000	0.0	0.000	0.000000	NaN	NaN	NaN	0.000000	0.018182	0.000000
Poncho	0.000000	0.009709	NaN	NaN	0.015152	0.000000	0.000000	0.0	0.000	0.003906	NaN	NaN	NaN	0.000000	0.000000	0.000000
Robe	0.000000	0.000000	NaN	NaN	0.000000	0.000000	0.000000	0.0	0.000	0.003906	NaN	NaN	NaN	0.000000	0.000000	0.000000
Romper	0.000000	0.009709	NaN	NaN	0.000000	0.000000	0.000000	0.0	0.000	0.039062	NaN	NaN	NaN	0.000000	0.000000	0.000000
Shorts	0.000000	0.000000	NaN	NaN	0.000000	0.000000	0.000000	0.0	0.750	0.000000	NaN	NaN	NaN	0.000000	0.000000	0.000000
Skirt	0.000000	0.000000	NaN	NaN	0.000000	0.000000	0.000000	0.0	0.000	0.015625	NaN	NaN	NaN	0.000000	0.000000	0.037037
Sweater	0.038462	0.058252	NaN	NaN	0.136364	0.000000	0.000000	0.0	0.000	0.011719	NaN	NaN	NaN	0.090909	0.036364	0.000000
Sweatpants	0.000000	0.000000	NaN	NaN	0.000000	0.000000	0.000000	0.0	0.000	0.000000	NaN	NaN	NaN	0.000000	0.000000	0.074074
Sweatshorts	0.000000	0.000000	NaN	NaN	0.000000	0.000000	0.000000	0.0	0.000	0.000000	NaN	NaN	NaN	0.000000	0.000000	0.037037
Tank	0.038462	0.038835	NaN	NaN	0.015152	0.000000	0.000000	0.0	0.000	0.007812	NaN	NaN	NaN	0.000000	0.018182	0.000000
Tee	0.000000	0.194175	NaN	NaN	0.030303	0.000000	0.000000	0.0	0.000	0.039062	NaN	NaN	NaN	0.000000	0.000000	0.000000
Top	0.000000	0.106796	NaN	NaN	0.000000	0.000000	0.000000	0.0	0.000	0.007812	NaN	NaN	NaN	0.000000	0.000000	0.000000
Trunks	0.000000	0.000000	NaN	NaN	0.000000	0.000000	0.000000	0.0	0.000	0.000000	NaN	NaN	NaN	0.000000	0.000000	0.000000

Jeggings	Joggers	Jumpsuit	Kimono	Leggings	Parka	Poncho	Robe	Romper	Shorts	Skirt	Sweater	Sweatpants	Sweatshorts	Tank	Tee	Top	Trunks
NaN	0.000000	0.000	0.142857	0.000000	0.00	NaN	NaN	0.000000	0.000000	0.000000	0.000000	0.000000	NaN	0.000000	0.000000	0.000000	NaN
NaN	0.000000	0.000	0.000000	0.000000	0.00	NaN	NaN	0.064516	0.000000	0.018182	0.026667	0.000000	NaN	0.200000	0.086093	0.042553	NaN
NaN	0.000000	0.000	0.000000	0.000000	0.00	NaN	NaN	0.000000	0.000000	0.000000	0.000000	0.000000	NaN	0.000000	0.000000	0.000000	NaN
NaN	0.076923	0.000	0.000000	0.000000	0.00	NaN	NaN	0.000000	0.017241	0.000000	0.000000	0.000000	NaN	0.000000	0.000000	0.000000	NaN
NaN	0.000000	0.000	0.285714	0.000000	0.00	NaN	NaN	0.000000	0.000000	0.054545	0.133333	0.000000	NaN	0.016667	0.006623	0.000000	NaN
NaN	0.076923	0.000	0.000000	0.000000	0.00	NaN	NaN	0.000000	0.017241	0.018182	0.000000	0.058824	NaN	0.000000	0.000000	0.000000	NaN
NaN	0.000000	0.000	0.000000	0.000000	0.00	NaN	NaN	0.000000	0.000000	0.000000	0.000000	0.000000	NaN	0.000000	0.000000	0.000000	NaN
NaN	0.000000	0.000	0.000000	0.000000	0.00	NaN	NaN	0.000000	0.051724	0.000000	0.000000	0.000000	NaN	0.000000	0.000000	0.000000	NaN
NaN	0.000000	0.100	0.428571	0.000000	0.00	NaN	NaN	0.354839	0.000000	0.254545	0.040000	0.000000	NaN	0.100000	0.052980	0.000000	NaN
NaN	0.000000	0.000	0.142857	0.000000	0.00	NaN	NaN	0.032258	0.000000	0.000000	0.000000	0.000000	NaN	0.000000	0.000000	0.000000	NaN
NaN	0.000000	0.000	0.000000	0.038462	0.00	NaN	NaN	0.000000	0.000000	0.000000	0.000000	0.000000	NaN	0.000000	0.000000	0.000000	NaN
NaN	0.000000	0.000	0.000000	0.000000	0.00	NaN	NaN	0.000000	0.000000	0.000000	0.013333	0.000000	NaN	0.000000	0.006623	0.000000	NaN
NaN	0.000000	0.025	0.000000	0.000000	0.00	NaN	NaN	0.000000	0.000000	0.000000	0.053333	0.000000	NaN	0.000000	0.013245	0.000000	NaN
NaN	0.115385	0.000	0.000000	0.000000	0.00	NaN	NaN	0.000000	0.000000	0.000000	0.026667	0.000000	NaN	0.016667	0.000000	0.000000	NaN
NaN	0.038462	0.000	0.000000	0.038462	0.00	NaN	NaN	0.000000	0.000000	0.018182	0.000000	0.000000	NaN	0.000000	0.000000	0.000000	NaN
NaN	0.500000	0.025	0.000000	0.076923	0.00	NaN	NaN	0.000000	0.000000	0.000000	0.000000	0.352941	NaN	0.000000	0.000000	0.000000	NaN
NaN	0.000000	0.375	0.000000	0.115385	0.00	NaN	NaN	0.096774	0.000000	0.018182	0.000000	0.000000	NaN	0.016667	0.006623	0.000000	NaN
NaN	0.000000	0.000	0.571429	0.000000	0.00	NaN	NaN	0.000000	0.000000	0.000000	0.000000	0.000000	NaN	0.000000	0.000000	0.000000	NaN
NaN	0.115385	0.000	0.000000	0.576923	0.00	NaN	NaN	0.000000	0.000000	0.018182	0.013333	0.176471	NaN	0.000000	0.006623	0.000000	NaN
NaN	0.000000	0.000	0.000000	0.000000	0.25	NaN	NaN	0.000000	0.000000	0.000000	0.013333	0.000000	NaN	0.000000	0.000000	0.000000	NaN
NaN	0.000000	0.000	0.000000	0.000000	0.00	NaN	NaN	0.000000	0.000000	0.026667	0.000000	0.000000	NaN	0.000000	0.000000	0.000000	NaN
NaN	0.000000	0.000	0.000000	0.000000	0.00	NaN	NaN	0.032258	0.000000	0.000000	0.000000	0.000000	NaN	0.000000	0.000000	0.000000	NaN
NaN	0.000000	0.025	0.000000	0.000000	0.00	NaN	NaN	0.483871	0.000000	0.018182	0.000000	0.000000	NaN	0.033333	0.006623	0.000000	NaN
NaN	0.038462	0.000	0.000000	0.000000	0.00	NaN	NaN	0.000000	0.758621	0.109091	0.000000	0.058824	NaN	0.000000	0.000000	0.000000	NaN
NaN	0.000000	0.000	0.000000	0.000000	0.00	NaN	NaN	0.000000	0.155172	0.745455	0.000000	0.000000	NaN	0.000000	0.000000	0.000000	NaN
NaN	0.000000	0.000	0.000000	0.000000	0.00	NaN	NaN	0.000000	0.000000	0.018182	0.520000	0.000000	NaN	0.016667	0.079470	0.000000	NaN
NaN	0.192308	0.000	0.000000	0.038462	0.00	NaN	NaN	0.000000	0.017241	0.018182	0.000000	0.411765	NaN	0.000000	0.000000	0.000000	NaN
NaN	0.000000	0.000	0.000000	0.000000	0.00	NaN	NaN	0.000000	0.034483	0.018182	0.000000	0.058824	NaN	0.000000	0.000000	0.000000	NaN
NaN	0.000000	0.000	0.000000	0.000000	0.00	NaN	NaN	0.000000	0.017241	0.000000	0.000000	0.000000	NaN	0.533333	0.099338	0.063830	NaN
NaN	0.000000	0.000	0.142857	0.000000	0.00	NaN	NaN	0.000000	0.000000	0.036364	0.093333	0.000000	NaN	0.283333	0.576159	0.196383	NaN
NaN	0.000000	0.000	0.000000	0.000000	0.00	NaN	NaN	0.000000	0.000000	0.036364	0.040000	0.000000	NaN	0.166667	0.079470	0.148936	NaN
NaN	0.000000	0.000	0.000000	0.000000	0.00	NaN	NaN	0.000000	0.034483	0.000000	0.000000	0.000000	NaN	0.000000	0.000000	0.000000	NaN

Appendix B: Snippets of the Github Repository



```
+accuracy_confusionMatrix
  -accuracy_confusionMatrix.py
+data
  -label_map file
  -train TFRecord file
  -test TFRecord file
  -eval TFRecord file
+docs
+generate_tfrecord
  -deep_fashion_to_tfrecord.py
+inference
  -Object_detection_image.py
  -Object_detection_webcam.py
+models
  + <model name>
    -pipeline config file
    +train
    +eval
+tf
  +models (tensorflow models repository)
-requirements.txt
```

Preparing Environment Variables

```
export PYTHONPATH=$PYTHONPATH:<path to project directory>/tf/models/research:<path to project directory>/t
```

Remember to run this in every terminal session prior to doing anything else. Or you can put this in your .bashrc (.bash_profile for macOS) file for persistency.

Dataset Preparation

```
# From the project root directory
DATASET_PATH={Path to DeepFashion project dataset with Anno, Eval and Img directories e.g. /home/user/deep
OUTPUT_PATH={Path to output TFRecord e.g. data/val.record}
CATEGORIES={broad or fine, broad FOR top, bottom or full only, fine FOR categories.}
EVALUATION_STATUS={train, val or Test}
LABEL_MAP_PATH={Path to label map proto, e.g. data/deepfashion_label_map_fine.pbtxt.}

python generate_tfrecord/deep_fashion_to_tfrecord.py \
    --dataset_path ${DATASET_PATH} \
    --output_path ${OUTPUT_PATH} \
    --categories ${CATEGORIES} \
    --evaluation_status ${EVALUATION_STATUS} \
    --label_map_path ${LABEL_MAP_PATH}
```

Training and Evaluation

1. Training and evaluation using the sample configs:

```
# From the tensorflow/models/research/ directory
cd tf/models/research/

PIPELINE_CONFIG_PATH={path to pipeline config file e.g. models/<model name>/pipeline.config}
MODEL_DIR={path to model directory e.g. models/<model name>/}
NUM_TRAIN_STEPS={60000 was used FOR SsdResnet50, 3000000 was used FOR fasterRcnnInceptionV2}
SAMPLE_1_OF_N_EVAL_EXAMPLES=50

python object_detection/model_main.py \
    --pipeline_config_path=${PIPELINE_CONFIG_PATH} \
    --model_dir=${MODEL_DIR} \
    --num_train_steps=${NUM_TRAIN_STEPS} \
    --sample_1_of_n_eval_examples=$SAMPLE_1_OF_N_EVAL_EXAMPLES \
    --alsologtostderr
```

2. Running Tensorboard:

```
MODEL_DIR={path to model directory e.g. models/<model name>/}
tensorboard --logdir=${MODEL_DIR}
```

Exporting a Trained Model

```
# From tensorflow/models/research/
cd tf/models/research/

INPUT_TYPE=image_tensor
PIPELINE_CONFIG_PATH={path to pipeline config file e.g. models/<model name>/pipeline.config}
TRAINED_CKPT_PREFIX={path to model.ckpt e.g. models/<model name>/model.ckpt-<CHECKPOINT_NUMBER>}
EXPORT_DIR={path to folder that will be used For Export e.g. models/<model name>/inference_graph }

python object_detection/export_inference_graph.py \
    --input_type=${INPUT_TYPE} \
    --pipeline_config_path=${PIPELINE_CONFIG_PATH} \
    --trained_checkpoint_prefix=${TRAINED_CKPT_PREFIX} \
    --output_directory=${EXPORT_DIR}
```

Generating Top-N Accuracy and Confusion Matrix

```
# From the project root directory

CKPT_PATH={Path to frozen detection graph .pb file, Which contains the model that is used e.g. models/<mod
LABEL_MAP_PATH={Path to label map proto, e.g. data/deepfashion_label_map_fine.pbtxt.}
NUM_CLASSESS={Number of classes the labels have. e.g. 50 FOR DeepFashion}
TFRECORD_PATH={Should use evaluation Set .e.g data/eval.record}
SKIP_EVERY={Same as sample_1_of_n_eval_examples, skip every n images during evaluation. e.g. 50 was used I
TOP={Top N accuracy needed. e.g. 1 or 3 were used In the paper}
MINIMUM_SCORE={Threshold FOR minimum score that is considered a True positive. e.g. 0.3}
OUTPUT_RESULT={Path to a text file containing all the results}
CONFUSION_TOPN={True or False FOR whether the confusion matrix should follow top N results.}

python accuracy_confusionMatrix/accuracy_confusionMatrix.py \
    --ckpt_path ${CKPT_PATH} \
    --label_map_path ${LABEL_MAP_PATH} \
    --num_classes ${NUM_CLASSESS} \
    --tfrecord_path ${TFRECORD_PATH} \
    --skip_every ${SKIP_EVERY} \
    --top ${TOP} \
    --minimumScore ${MINIMUM_SCORE} \
    --outputResult ${OUTPUT_RESULT} \
    --confusion_topn ${CONFUSION_TOPN}
```

Inferencing on Webcam

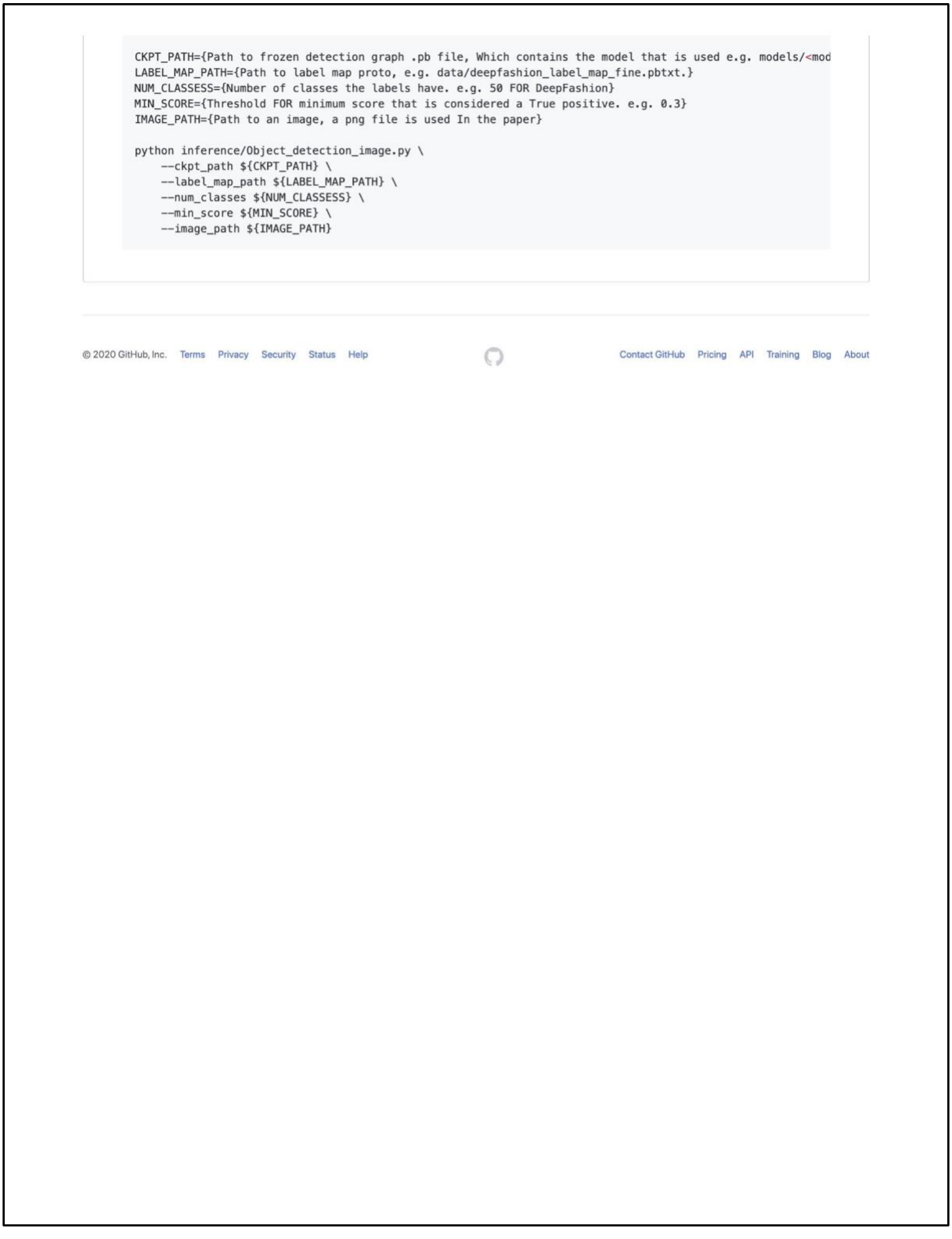
```
# From the project root directory

CKPT_PATH={Path to frozen detection graph .pb file, Which contains the model that is used e.g. models/<mod
LABEL_MAP_PATH={Path to label map proto, e.g. data/deepfashion_label_map_fine.pbtxt.}
NUM_CLASSESS={Number of classes the labels have. e.g. 50 FOR DeepFashion}
MIN_SCORE={Threshold FOR minimum score that is considered a True positive. e.g. 0.3}

python inference/Object_detection_webcam.py \
    --ckpt_path ${CKPT_PATH} \
    --label_map_path ${LABEL_MAP_PATH} \
    --num_classes ${NUM_CLASSESS} \
    --min_score ${MIN_SCORE}
```

Inferencing on an Image

```
# From the project root directory
```



```
CKPT_PATH={Path to frozen detection graph .pb file, Which contains the model that is used e.g. models/<mod
LABEL_MAP_PATH={Path to label map proto, e.g. data/deepfashion_label_map_fine.pbtxt.}
NUM_CLASSESS={Number of classes the labels have. e.g. 50 FOR DeepFashion}
MIN_SCORE={Threshold FOR minimum score that is considered a True positive. e.g. 0.3}
IMAGE_PATH={Path to an image, a png file is used In the paper}

python inference/Object_detection_image.py \
    --ckpt_path ${CKPT_PATH} \
    --label_map_path ${LABEL_MAP_PATH} \
    --num_classes ${NUM_CLASSESS} \
    --min_score ${MIN_SCORE} \
    --image_path ${IMAGE_PATH}
```

© 2020 GitHub, Inc. Terms Privacy Security Status Help Contact GitHub Pricing API Training Blog About

This page is intentionally left blank to indicate the back cover. Ensure that the back cover is black in color.